INSIDE MACINTOSH 系列丛书

苹果电脑有限公司

INSIDE MACINTOSH
工具箱基础

科学出版社

INSIDE MACINTOSH系列丛书

INSIDE MACINTOSH
工具箱基础

苹果电脑有限公司

科学出版社

1997

内 容 简 介

本书介绍了Macintosh应用程序实质性部分及可以使用执行的系统软件程序。书中不仅介绍了事件、窗口、菜单、控制、警告栏和对话框，而且还讨论了应用程序如何与Finder交互作用等问题。书中介绍的系统软件例程在使用pascal、C或汇编语言界面时均可使用，其列出的所有代码段均采用pascal表示并介绍了使用各种例程的方法、使用图解描述了完成特定任务的技术。书末还对Finder界面做了详述。

本书可供计算机应用与开发人员、大专院校师生参考。

版 权 声 明

本书英文原版由苹果电脑有限公司授权Addison-Wesley出版公司出版。本书汉译本由苹果电脑有限公司授予科学出版社独家出版，并限于中国大陆发行。版权归苹果电脑有限公司所有。未经出版者书面许可，本书的任何部分均不得以任何形式或任何手段复制或传播。

图书在版编目（CIP）数据

Inside Macintosh 工具箱基础/苹果电脑有限公司著 - 北京：
科学出版社，1996.7
(Inside Macintosh系列从书)
ISBN 7-03-004983-7

I. I... II. 美... III. 微型计算机—软件工具 IV. TP311.56

中国版本图书馆CIP数据核字(96)第10193号

科 学 出 版 社

北京东黄城根北街16号

邮政编码：100717

中国科学院印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1997年5月第一版 开本：850×1168 1/16
1997年5月第一次印刷 印张：59 3/4
印数：1—3000 字数：1 100 000

定价：95.00元

目　　录

前　言	**关于本书**	**xxv**
	典型章节的格式	xxvi
	本书中使用的一些惯例和规则	xxvi
	特殊字体	xxvii
	附注的类型	xxvii
	空字符串	xxvii
	汇编语言信息	xxvii
	开发环境	xxviii
第一章	**Macintosh工具箱引论**	**1-1**
	Macintosh 工具箱概述	1-4
	事件	1-5
	菜单	1-6
	窗口	1-6
	控制	1-7
	警告栏和对话框	1-8
	图标及其它和Finder的交互作用	1-10
	资源	1-11
	旁述	1-14
	拷贝和粘贴	1-14
	相关的系统软件功能	1-14
	在屏幕上绘图	1-14
	处理文字	1-14
	管理文件	1-15
	分配存储器和启用进程	1-15
	创建版本和引用程序	1-15
	和其它应用程序通信	1-16
	设计您的应用程序	1-16
第二章	**事件管理器**	**2-1**
	事件介绍	2-4
	低级事件	2-8
	操作系统事件	2-10
	高级事件	2-13
	事件优先级	2-15
	转换上下文	2-15

关于事件管理器 2-16
事件管理器的使用 2-17
获取关于事件的信息 2-18
处理事件 2-21
调用 WaitNextEvent 函数 2-22
事件循环的编写 2-24
设定事件类型屏蔽参数 2-26
处理对话框中的事件 2-29
创建'SIZE'资源 2-30
处理低级事件 2-32
响应鼠标事件 2-33
响应键盘事件 2-38
扫描取消事件 2-46
响应更新事件 2-47
响应激活事件 2-50
响应磁盘插入事件 2-55
响应空事件 2-57
处理操作系统事件 2-58
响应暂停和恢复执行事件 2-60
响应鼠标移动事件 2-62
处理高级事件 2-67
响应来自其它应用程序的事件 2-69
搜索指定的高级事件 2-71
确定高级事件的发送程序 2-72
发送高级事件 2-73
请求回执 2-77
处理苹果事件 2-78
事件管理器参考资料 2-78
数据结构 2-79
事件记录 2-79
目标标识符记录 2-81
高级事件消息记录 2-82
事件队列 2-83
事件管理器例程 2-84
接收事件 2-84
发送事件 2-100
转换进程序号和端口名称 2-105
读鼠标 2-108
读键盘 2-110
获取计时信息 2-112
应用程序定义的例程 2-114
用于搜索高级事件队列的过滤函数 2-114
资源 2-115
'SIZE'资源 2-115
事件管理器总结 2-120
Pascal 小结 2-120
常数 2-120

数据类型 2-122
事件管理器例程 2-123
应用程序定义的例程 2-124
C 小结 2-125
常数 2-125
数据类型 2-127
事件管理器例程 2-128
应用程序定义的例程 2-129
汇编语言小结 2-130
数据结构 2-130
陷阱宏 2-130
全局变量 2-131
结果代码 2-132

第三章 菜单管理器 3-1

菜单介绍 3-5
菜单和菜单栏定义例程 3-9
菜单栏 3-9
菜单 3-11
菜单命令项目 3-12
菜单命令项目分组 3-14
菜单命令的等效命令组合键 3-16
菜单管理器自动添加菜单 3-19
苹果菜单 3-20
文件菜单 3-22
编辑菜单 3-24
字体菜单 3-26
大小菜单 3-27
帮助菜单 3-29
键盘菜单 3-32
应用程序菜单 3-33
弹出式菜单 3-33
嵌套式菜单 3-38
关于菜单管理器 3-39
菜单管理器如何保留有关菜单的信息 3-40
菜单管理器如何保留有关应用程序菜单栏的信息 3-40
使用菜单管理器 3-41
创建菜单 3-42
创建菜单资源 3-43
创建菜单栏资源 3-49
设置应用程序的菜单栏 3-50
创建嵌套式菜单 3-53
创建弹出式菜单 3-56

更改菜单中的项目外观 3-57
启用和停用菜单命令项目 3-58
更改项目文字 3-59
更改菜单命令项目的字形 3-60
更改菜单命令项目的标记 3-61
更改菜单命令项目的图标或语系输入代码 3-62
添加菜单命令项目 3-64
在帮助菜单中添加项目 3-67
在苹果菜单中添加项目 3-68
添加字体菜单 3-69
处理用户选择的菜单命令 3-70
处理菜单栏中的鼠标按下事件 3-72
调整应用程序菜单 3-73
确定用户是否选择了等效命令组合键 3-77
响应用户选择的菜单命令项目 3-78
处理苹果菜单 3-80
处理帮助菜单 3-81
处理大小菜单 3-82
从对话框中存取菜单 3-84
编写您自己的菜单定义过程 3-87
计算菜单的尺寸 3-89
绘制菜单中的菜单命令项目 3-90
确定光标是否位于启用菜单命令项目 3-92
菜单管理器参考资料 3-95
数据结构 3-95
菜单记录 3-95
菜单列表 3-97
菜单色彩信息表记录 3-98
菜单管理器例程 3-102
初始化菜单管理器 3-103
创建菜单 3-105
向或从当前菜单列表中添加或删除菜单 3-108
从'MBAR'资源中获取菜单栏说明信息 3-110
获取和设定菜单栏 3-112
绘制菜单栏 3-113
响应用户选择的菜单命令 3-114
为菜单记录获取一个间接指针 3-122
添加和删除菜单命令项目 3-124
获取和设定菜单命令项目的外观 3-130
排列菜单 3-140
计算菜单中的项目数 3-140
醒目显示菜单栏 3-141
重新计算菜单尺寸 3-142
管理菜单色彩信息表中的入口 3-143
应用程序定义的例程 3-148
菜单定义过程 3-148

资源 3-151
菜单资源 3-151
菜单栏资源 3-155
菜单色彩信息表资源 3-155
菜单定义过程资源 3-157
菜单管理器小结 3-158
Pascal 小结 3-158
常数 3-158
数据类型 3-158
菜单管理器例程 3-159
应用程序定义的例程 3-162
C 小结 3-162
常数 3-162
数据类型 3-163
菜单管理器例程 3-164
应用程序定义的例程 3-166
汇编语言小结 3-167
数据结构 3-167
全局变量 3-167
结果代码 3-167

第四章 窗口管理器 4-1

介绍窗口 4-4
当前活动窗口和非当前活动窗口 4-6
窗口类型 4-8
窗口区域 4-12
对话框和警告栏 4-13
控制部件 4-14
桌面环境中的窗口 4-15
关于窗口管理器 4-16
图形端口 4-17
窗口记录 4-19
彩色窗口 4-20
窗口中的事件 4-21
使用窗口管理器 4-22
管理多窗口 4-23
创建一个窗口 4-25
定义窗口资源 4-25
凭借资源创建一个窗口 4-27
在桌面环境中对文稿窗口定位 4-30
绘制窗口内容 4-39
更新内容区域 4-40
保留更新区域 4-41
处理窗口中的事件 4-41
处理窗口中的鼠标事件 4-42

处理窗口中的键盘事件 4-47
处理更新事件 4-48
处理激活事件 4-50
移动窗口 4-53
缩放窗口 4-53
调整窗口大小 4-57
关闭窗口 4-60
隐藏和显示窗口 4-62
窗口管理器参考资料 4-64
数据结构 4-65
彩色窗口记录 4-65
窗口记录 4-69
窗口状态数据记录 4-70
窗口彩色表记录 4-71
辅助窗口记录 4-73
窗口列表 4-74
窗口管理器例程 4-74
初始化窗口管理器 4-74
命名窗口 4-85
显示窗口 4-86
检索窗口信息 4-91
移动窗口 4-94
调整窗口大小 4-99
缩放窗口 4-101
关闭窗口和解除窗口分配 4-103
保留更新区域 4-106
设定和检索其它窗口特性 4-109
操纵桌面环境 4-112
操纵窗口彩色信息 4-114
低级例程 4-116
应用程序定义的例程 4-120
窗口定义函数 4-120
资源 4-124
窗口资源 4-124
窗口定义函数资源 4-127
窗口彩色表资源 4-127
窗口管理器总结 4-129
Pascal 概述 4-129
常数 4-129
数据类型 4-131
窗口管理器例程 4-133
应用程 序定义的例程 4-135
C 概述 4-136
常数 4-136
数据类型 4-138

窗口管理器例程 4-139
应用程序定义的例程 4-142
汇编语言概述 4-143
数据类型 4-143
全局变量 4-144

第五章 控制管理器 5-1

控制部件介绍 5-4
按钮 5-5
复选框 5-5
选项按钮 5-6
弹出式菜单 5-6
滚动杆 5-7
其它控制部件 5-11
有效和无效控制 5-11
控制定义函数 5-14
关于控制管理器 5-14
使用控制管理器 5-15
创建和显示控制部件 5-15
创建按钮、复选框或选项按钮 5-17
创建滚动杆 5-21
创建弹出式菜单 5-25
更新控制部件 5-29
响应控制部件中的鼠标事件 5-30
确定控制部件中的鼠标按下事件 5-31
跟踪控制部件中的光标 5-35
确定和更改控制部件设定状态 5-37
滚动文稿 5-43
为响应滚动杆中的事件进行滚动 5-53
为响应滚动箭头和灰度面积中的事件进行滚动 5-57
在窗口内绘制滚动后的文稿 5-62
移动和调整滚动杆大小 5-65
定义您自己的控制部件定义函数 5-71
控制管理器参考资料 5-72
数据结构 5-72
控制记录 5-73
辅助控制记录 5-76
弹出式菜单专用数据记录 5-77
控制部件色彩表记录 5-77
控制管理器例程 5-80
创建控制部件 5-81
绘制控制部件 5-85
处理控制部件中的鼠标事件 5-88
更改控制设定值和显示 5-93
确定控制部件的值 5-102
取消控制部件 5-108

应用程序定义的例程 5-109
定义您自己的控制部件定义函数 5-109
定义您自己的动作过程 5-115
资源 5-117
控制资源 5-118
控制部件色彩表资源 5-121
控制部件定义函数 5-123
控制管理器总结 5-124
Pascal 小结 5-124
常数 5-124
数据类型 5-126
控制管理器例程 5-127
应用程序定义的例程 5-129
C 小结 5-129
常数 5-129
数据类型 5-131
控制管理器例程 5-132
应用程序定义的例程 5-134
汇编语言总结 5-134
数据结构 5-134
全局变量 5-135

第六章 对话管理器 6-1

警告和对话框介绍 6-6
警告类型 6-8
对话框类型 6-9
定模式对话框 6-10
可移动定模式对话框 6-11
不定模式对话框 6-12
警告和对话框中的项目 6-13
警告和对话框中的事件 6-14
警告栏、对话框和窗口管理器 6-15
关于对话管理器 6-16
使用对话管理器 6-17
创建示警声和警告栏 6-18
创建对话框 6-23
为警告栏和对话框提供项目 6-26
项目类型 6-30
显示矩形 6-32
启用和停用项目 6-36
项目的资源标识符 6-36
按钮、复选框和选项按钮的标题 6-37
静态文本项目和可编辑文字项目的文字字符串 6-40
弹出式菜单命令项目 6-42
项目间键盘引导 6-44

操纵项目 6-44
更改静态文字 6-46
从可编辑文字项目中获取文字 6-48
在一个现有的对话框中添加项目 6-51
使用一个定义应用程序的项目来绘制一个系统设定按钮的黑体轮廓线 6-56
显示警告栏和对话框 6-61
定位警告栏和对话框 6-62
停用警告栏和定模式对话框后面的窗口 6-64
显示不定模式对话框 6-66
针对定模式对话框调整菜单 6-68
针对可移动定模式和不定模式对话框调整菜单 6-73
显示多个警告栏和对话框 6-74
从后台显示警告栏和对话框 6-74
在您的警告栏和对话框中加入彩色 6-75
处理警告栏和对话框中的事件 6-77
响应控制部件中的事件 6-78
响应可编辑文字项目中的事件 6-79
响应警告栏中的事件 6-81
响应定模式对话框中的事件 6-82
为警告栏和定模式对话框编写事件过滤函数 6-86
响应不定模式和可移动定模式对话框中的鼠标事件 6-89
响应不定模式和可移动定模式对话框中的键盘事件 6-94
响应不定模式和可移动定模式对话框中的激活和更新事件 6-97
关闭对话框 6-100
对话管理器参考资料 6-101
数据结构 6-101
对话记录 6-101
对话管理器例程 6-102
初始化对话管理器 6-102
创建警告 6-105
创建和处理对话框 6-113
操纵警告栏和对话框中的项目 6-120
处理警告栏和对话框中的文字 6-129
处理对话框中的事件 6-135
应用程序定义的例程 6-143
资源 6-147
对话资源 6-148
警告资源 6-150
项目列表资源 6-151
对话项目色彩表资源 6-156
警告色彩表资源 6-157
项目色彩表资源 6-158

对话管理器小结 6-165
Pascal 小结 6-165
常数 6-165
数据类型 6-166
对话管理器例程 6-166
应用程序定义的例程 6-168
C 小结 6-168
常数 6-168
数据类型 6-169
对话管理器例程 6-170
应用程序定义的例程 6-172
汇编语言小结 6-172
数据结构 6-172
全局变量 6-172

第七章 Finder 界面 7-1

Finder 界面介绍 7-3
关于 Finder 界面 7-6
使用 Finder 界面 7-6
给您的应用程序署名，给您的文稿设定创建者和文件类型 7-8
为 Finder 创建图标 7-11
创建自定的文稿图标 7-17
创建文件参考资源 7-18
创建束资源 7-20
Finder 打开您的应用程序的方式和时间 7-25
Finder 找不到您的应用程序时显示消息 7-27
提供版本资源 7-31
使用目录文件中的 Finder 信息 7-32
支持固定样板文稿 7-34
分配字体、语音及其它可移动资源 7-36
为非文稿图标提供旁述 7-38
使用替身 7-39
使用系统文件夹及和它有关的目录 7-41
桌面数据库 7-45
Finder 界面参考资料 7-46
数据结构 7-46
文件信息记录 7-47
扩展的文件信息记录 7-49
目录信息记录 7-50
扩展的目录信息记录 7-50
例程 7-51
分辨替身文件 7-51
查找目录 7-53

资源 7-56
署名资源 7-57
图标列表资源 7-57
小图标列表资源 7-58
大 4 位彩色图标记录 7-59
小 4 位彩色图标记录 7-60
大 8 位彩色图标记录 7-61
小 8 位彩色图标记录 7-62
图标资源 7-63
彩色图标资源 7-64
文件参考资源 7-64
束资源 7-65
未找到应用程序名称字符串 7-68
未找到应用程序消息字符串 7-68
版本资源 7-69
Finder 界面小结 7-71
Pascal 小结 7-71
常数 7-71
数据类型 7-73
例程 7-74
C 小结 7-74
常数 7-74
数据类型 7-76
例程 7-77
汇编语言小结 7-77
数据结构 7-77
结果代码 7-78

词汇表 GL-1

索引 IN-1

图形、表格和列表

彩页

彩页附在标题页后面

彩图 1 一个彩色窗口
彩图 2 一个彩色警告栏
彩图 3 一个图标库
彩图 4 文稿图标

第一章 **Macintosh 工具箱引论** 1-1

图 1-1 在桌面环境下拥有多个窗口的 SurfWriter 应用程序 1-3
图 1-2 一个典型的窗口 1-6
图 1-3 一般控制部件 1-7
图 1-4 一个警告栏 1-8
图 1-5 定模式对话框、可移动定模式对话框及不定模式对话框 1-9

第二章 **事件管理器** 2-1

图 2-1 事件来源 2-6
图 2-2 低级事件 2-10
图 2-3 操作系统事件 2-11
图 2-4 高级事件 2-14
图 2-5 事件记录的 modifiers 域 2-20
图 2-6 事件类型屏蔽参数 2-27
图 2-7 用于键盘事件的事件记录中的Message 域 2-40
图 2-8 键盘转换 2-41
图 2-9 苹果键盘 II，ISO 布局的模拟键代码 2-42
图 2-10 苹果扩充键盘II模拟键代码 2-43
图 2-11 响应窗口更新事件 2-49
图 2-12 响应窗口激活事件 2-52
图 2-13 标准的箭头光标 2-63
图 2-14 I 型光标、十字准线光标、加号光标和手表光标 2-63
图 2-15 箭头光标区域和I型光标区域 2-64
图 2-16 将光标从I型光标改变为箭头光标 2-65
图 2-17 KeyTranslate 函数结果的结构 2-111

程序清单 2-1 使用 WaitNextEvent 函数 2-23
程序清单 2-2 事件循环 2-24
程序清单 2-3 处理事件 2-26
程序清单 2-4 示例 SIZE’资源的 Rez 输入 2-31
程序清单 2-5 处理鼠标按下事件 2-34
程序清单 2-6 处理键按下和自动键事件 2-44
程序清单 2-7 处理键按下事件 2-44
程序清单 2-8 扫描“Command”+“.”组合键事件 2-46

程序清单 2-9　响应更新事件　2-50
程序清单 2-10　响应活跃事件　2-53
程序清单 2-11　响应磁盘插入事件　2-56
程序清单 2-12　处理空事件　2-57
程序清单 2-13　响应操作系统事件　2-59
程序清单 2-14　响应挂起和恢复执行事件　2-61
程序清单 2-15　改变光标　2-65
程序清单 2-16　接受一个高级事件　2-70
程序清单 2-17　高级事件的发送程序　2-74
程序清单 2-18　使用 PPCBrowser 函数发送一个高级事件　2-76
程序清单 2-19　SIZE' 资源的 Rez 属性单元　2-116

第三章　菜单管理器　3-1

图 3-1　下拉式菜单、子菜单和弹出式菜单　3-6
图 3-2　显示编辑菜单的SurfWriter应用程序菜单栏　3-7
图 3-3　SurfWriter应用程序的菜单栏　3-10
图 3-4　用另一种语系系统本地化的SurfWriter应用程序的菜单栏　3-10
图 3-5　两个带有各种特征的菜单　3-13
图 3-6　在一个互斥分组中的菜单命令项目　3-15
图 3-7　在一个相容分组中的菜单命令项目　3-15
图 3-8　在一个相容分组中使用选择标记和破折号　3-15
图 3-9　SurfWriter 应用程序的苹果菜单　3-21
图 3-10　选择 SurfWriter 应用程序的“关于...”命令　3-22
图 3-11　一个应用程序的标准文件菜单　3-22
图 3-12　一个应用程序的标准编辑菜单　3-24
图 3-13　一个典型的字体菜单　3-26
图 3-14　一个显示了选定项包含一种以上字体的字体菜单　3-27
图 3-15　一个典型的大小菜单　3-28
图 3-16　一个选择新点数字体的对话框　3-28
图 3-17　输入一个新的字体大小　3-29
图 3-18　带有一种添加字体大小的“其它”命令　3-29
图 3-19　SurfWriter 应用程序的帮助菜单　3-30
图 3-20　苹果菜单和应用程序菜单的系统设定旁述　3-31
图 3-21　不同状态时“剪”命令的旁述　3-31
图 3-22　在一个应用程序中存取键盘菜单　3-32
图 3-23　SurfWriter 的应用程序菜单　3-33
图 3-24　一个弹出式菜单　3-34
图 3-25　一个弹出式菜单的关闭和打开状态　3-34
图 3-26　在弹出式菜单中进行选择　3-35
图 3-27　从多种属性的列表中选择一种属性　3-36
图 3-28　一个带有复选框和弹出式菜单的对话框　3-37
图 3-29　一个典型的关闭和打开状态下的弹出式菜单　3-37
图 3-30　具有添加的用户选择的键入弹出式菜单　3-38
图 3-31　一个嵌套式菜单命令项目和它的子菜单　3-39
图 3-32　带有子菜单的菜单命令项目　3-53

图 3-33	对话框中的弹出式菜单	3-56
图 3-34	菜单命令项目中的图标	3-63
图 3-35	添加了用户指定的字体大小的大小菜单	3-82
图 3-36	从定模式对话框中存取菜单	3-85
图 3-37	经编译的菜单（'MENU'）资源结构	3-152
图 3-38	说明菜单命令项目的可变长度数据， 由标准菜单定义过程定义	3-153
图 3-39	经编译的菜单栏（'MBAR'）资源的结构	3-155
图 3-40	经编译的菜单色彩信息表（'mctb'）资源的结构	3-156
图 3-41	在'mctb'资源中一个菜单彩色输入项的结构	3-157
表 3-1	为所有系统保留的等效命令组合键	3-18
表 3-2	为世界范围的系统保留的等效命令组合键	3-19
表 3-3	其它常用的等效命令组合键	3-19
表 3-4	响应标准文件菜单命令的动作	3-23
表 3-5	针对标准编辑菜单命令的动作	3-25
表 3-6	指定菜单资源中一个菜单命令项目的子菜单、 语系代码、缩小图标、小图标和彩色图标	3-46
表 3-7	菜单栏输入的彩色信息	3-100
表 3-8	菜单管理器例程新旧名称对照	3-102
程序清单 3-1	苹果菜单 'MENU' 资源的 Rez 输入	3-43
程序清单 3-2	一个编辑菜单 'MENU' 资源的 Rez 输入	3-48
程序清单 3-3	一个文件菜单 'MENU' 资源的 Rez 输入	3-49
程序清单 3-4	'MBAR' 资源的 Rez 输入	3-49
程序清单 3-5	设定一个应用程序的菜单和菜单栏	3-50
程序清单 3-6	存储和还原菜单彩色信息	3-52
程序清单 3-7	带有子菜单的嵌套式菜单的 Rez 输入	3-54
程序清单 3-8	创建嵌套式菜单	3-55
程序清单 3-9	更改菜单命令项目的文字	3-59
程序清单 3-10	设定菜单命令项目的字形	3-60
程序清单 3-11	添加菜单命令项目标记和删除菜单命令项目标记	3-61
程序清单 3-12	指定菜单命令项目图标	3-63
程序清单 3-13	菜单命令项目文字的 Rez 输入	3-66
程序清单 3-14	给帮助菜单添加项目	3-68
程序清单 3-15	给苹果菜单增加菜单命令项目	3-69
程序清单 3-16	为菜单添加字体名称	3-70
程序清单 3-17	确定一个鼠标按下事件是否发生	3-72
程序清单 3-18	判断光标何时在菜单栏中	3-72
程序清单 3-19	调整一个应用程序菜单	3-74
程序清单 3-20	为一个文稿窗口调整文件菜单	3-74
程序清单 3-21	为一个文稿窗口调整编辑菜单	3-75
程序清单 3-22	确定键是什么时候被按下的	3-77
程序清单 3-23	检验一个键按下事件是不是一个等效命令组合键	3-78
程序清单 3-24	响应用户的菜单命令选择	3-79
程序清单 3-25	响应用户在苹果菜单对一个项目的选择	3-80
程序清单 3-26	用户从帮助菜单中选择命令予以响应	3-81

程序清单 3-27	处理大小菜单	3-83
程序清单 3-28	一个示例菜单定义过程	3-89
程序清单 3-29	计算一个菜单的大小	3-90
程序清单 3-30	绘制菜单命令项目	3-91
程序清单 3-31	选择菜单命令项目	3-93

第四章 **窗口管理器** 4-1

图 4-1	桌面环境中的多个窗口	4-4
图 4-2	文稿窗口	4-5
图 4-3	当前活动和失活文稿窗口	4-7
图 4-4	一个 ZoomDocProc 类型的窗口	4-8
图 4-5	ZoomDocProc 类型的窗口，带有尺寸格和滚动杆	4-9
图 4-6	警告栏和固定位置定模式对话框的窗口类型 4-9	
图 4-7	movableDBoxProc 类型的窗口	4-10
图 4-8	noGrowDocProc 类型的窗口	4-10
图 4-9	不常用的窗口类型	4-11
图 4-10	窗口框架、内容区域和结构区	4-12
图 4-11	滚动杆	4-14
图 4-12	对话框中的控制部件	4-15
图 4-13	QuickDraw 全局坐标平面	4-17
图 4-14	窗口的局部和全局坐标系统	4-19
图 4-15	单个屏幕上的文稿窗口位置	4-31
图 4-16	“填充”空出的文稿窗口位置	4-31
图 4-17	多个屏幕上的文稿窗口位置	4-33
图 4-18	某个窗口的移动造成另一个窗口的更新区域的变动	4-40
图 4-19	醒目显示的关闭格和取消醒目显示的关闭格	4-46
图 4-20	醒目显示的缩放格及取消醒目显示的缩放格	4-47
图 4-21	BeginUpdate 和 EndUpdate 对可视区域及更新区域的影响	4-49
图 4-22	HideWindow、ShowWindow 和 SelectWindow 对窗口的累计影响	4-63
图 4-23	DragGrayRgn 使用的限制矩形	4-98
图 4-24	编译的窗口（‘WIND’）资源的结构	4-124
图 4-25	经过编译的窗口色彩表（‘wctb’）资源的结构	4-128

程序清单 4-1	定义窗口类型	4-25
程序清单 4-2	文稿窗口的窗口（‘WIND’）资源的 Rez 输入	4-26
程序清单 4-3	创建一个新窗口	4-28
程序清单 4-4	用于储存窗口状态数据的应用程序定义的数据结构	4-34
程序清单 4-5	存储一个文稿窗口的位置	4-34
程序清单 4-6	当用户打开一个存储的文稿时对窗口定位	4-36
程序清单 4-7	打开一个存储的文稿	4-37
程序清单 4-8	绘制一个窗口	4-39

程序清单 4-9 处理鼠标按下事件 4-44
程序清单 4-10 处理更新事件 4-50
程序清单 4-11 处理激活事件 4-51
程序清单 4-12 缩放窗口 4-55
程序清单 4-13 调整窗口大小 4-58
程序清单 4-14 当调整窗口大小时调整滚动杆和内容区域 4-59
程序清单 4-15 将窗口区域转换为局部坐标 4-60
程序清单 4-16 处理关闭命令 4-60
程序清单 4-17 关闭一个文稿 4-61
程序清单 4-18 显示隐藏的对话框 4-64

第五章 **控制管理器** 5-1

图 5-1 控制管理器提供的标准控制部件 5-4
图 5-2 一个系统设定按钮 5-5
图 5-3 一个选定的复选框 5-5
图 5-4 垂直滚动杆 5-7
图 5-5 使用滚动块和滚动箭头 5-8
图 5-6 文稿与窗口的空间关系，以及用滚动杆表示的方法 5-10
图 5-7 自定的滑块控制部件 5-11
图 5-8 对于用户选定的有效控制部件所作的可视反馈 5-12
图 5-9 无效控制部件 5-13
图 5-10 一个简单窗口中的按钮 5-17
图 5-11 简单窗口中的选项按钮 5-20
图 5-12 滚动杆如何和窗口框架重叠 5-22
图 5-13 一个弹出式菜单 5-26
图 5-14 示例弹出式菜单的尺寸 5-26
图 5-15 一个窗口中的 3 个控制部件 5-34
图 5-16 相对窗口移动文稿 5-46
图 5-17 更新经滚动的窗口的内容 5-49
图 5-18 窗口起始位置还原到（0, 0） 5-50
图 5-19 滚动到文稿结尾部分 5-51
图 5-20 更新窗口的内容和将窗口起始位置返回到（0, 0） 5-51
图 5-21 移动滚动杆和调整它的大小 5-66
图 5-22 调整大小后的窗口内的滚动杆，此时应用程序还没有移动该滚动杆 5-69
图 5-23 应用程序移动其左上点后的垂直滚动杆 5-69
图 5-24 自定控制部件 5-71
图 5-25 编译的控制（'CNTL'）资源结构 5-118
图 5-26 经过编译的控制部件色彩表（'cctb'）资源的结构 5-122

表 5-1 控制管理器例程的新名称与旧名称之间的映射 5-80

程序清单 5-1 为窗口创建一个按钮 5-17
程序清单 5-2 控制资源的 Rez 输入 5-18

程序清单 5-3	选项按钮控制资源的 Rez 输入	5-21
程序清单 5-4	窗口资源和它的滚动杆资源的 Rez 输入	5-23
程序清单 5-5	创建带有滚动杆的窗口	5-24
程序清单 5-6	弹出式菜单的控制资源的 Rez 输入	5-26
程序清单 5-7	响应窗口的更新事件	5-29
程序清单 5-8	重新绘制更新区域中的控制部件	5-30
程序清单 5-9	检测窗口中的鼠标按下事件	5-32
程序清单 5-10	检测弹出式菜单和按钮上的鼠标按下事件	5-33
程序清单 5-11	对按钮使用 TrackControl 函数	5-36
程序清单 5-12	对弹出式菜单使用 TrackControl	5-37
程序清单 5-13	对点按复选框的响应	5-38
程序清单 5-14	调整滚动杆的设定状态和位置	5-39
程序清单 5-15	为滚动杆指派设定值	5-40
程序清单 5-16	调整滚动杆的最大和当前设定值	5-41
程序清单 5-17	使用 ScrollRect 滚动窗口中显示的各部分	5-47
程序清单 5-18	响应滚动杆中的鼠标事件	5-53
程序清单 5-19	用于滚动文字文稿的动作过程	5-59
程序清单 5-20	用动作过程移动滚动块	5-61
程序清单 5-21	一个应用程序定义的更新例程	5-62
程序清单 5-22	刷新包含有图形对象的窗口	5-63
程序清单 5-23	在滚动一个 TextEdit 编辑记录后刷新窗口	5-65
程序清单 5-24	改变窗口滚动杆的大小和位置	5-67

第六章 **对话管理器** 6-1

图 6-1	Finder 使用的一个警告栏	6-6
图 6-2	一个典型的对话框	6-7
图 6-3	“注意”警告	6-8
图 6-4	“当心”警告	6-9
图 6-5	“停止”警告	6-9
图 6-6	定模式对话框	6-10
图 6-7	可移动定模式对话框	6-11
图 6-8	不定模式对话框	6-12
图 6-9	对话框中的典型项目	6-13
图 6-10	告诫用户存储更改过的文稿的警告栏	6-19
图 6-11	仅在第三和第四警告阶段显示的一个警告栏	6-21
图 6-12	一个简单的定模式对话框	6-24
图 6-13	各种资源与警告栏的关系	6-27
图 6-14	在警告栏中的一个安全的系统设定按钮	6-31
图 6-15	在警告栏中按钮和文字的统一间距	6-33
图 6-16	经适当调整大小的备用语系系统显示矩形及未经适当调整的情形	6-35
图 6-17	无效控制按钮和停用项目	6-37
图 6-18	带有“确定”和“取消”按钮的一个对话框	6-38
图 6-19	带有一个“停止”按钮的可移动定模式对话框	6-39
图 6-20	带有一个“还原”按钮的一个警告栏	6-39
图 6-21	一条难于理解且无用的警告信息	6-41

图 6-22	一个稍微好理解一些的警告消息	6-41
图 6-23	一个清楚且有用的警告消息	6-41
图 6-24	在一个对话框中的弹出式菜单	6-42
图 6-25	选定的滚动列表	6-45
图 6-26	显示了一个文稿名称的警告栏	6-46
图 6-27	在一个不定模式对话框中的两个可编辑文字项目	6-48
图 6-28	一个现有的对话框和要增补的项目	6-51
图 6-29	重叠项目以后的对话框	6-52
图 6-30	将项目增补到右边以后的对话框	6-52
图 6-31	将项目增补到底部以后的对话框	6-53
图 6-32	在对话框中，相对于一个现有的项目增补了一个项目	6-53
图 6-33	在一个文稿窗口前面的警告栏	6-63
图 6-34	主屏幕上的警告栏	6-63
图 6-35	位于文稿窗口屏幕的警告位置的警告栏	6-64
图 6-36	只有在第三警告阶段之后才会显示的警告栏	6-65
图 6-37	用于在文稿中更改文字的不定模式对话框	6-66
图 6-38	显示一个定模式对话框时的菜单存取	6-69
图 6-39	CautionAlert 为其报告事件的 3 个按钮	6-81
图 6-40	ModalDialog 报告其事件的 4 个项目	6-83
图 6-41	DialogSelect 为其报告事件的一个不定模式对话框	6-91
图 6-42	编译了的对话（‘DLOG’）资源的结构	6-148
图 6-43	编译了的警告（'ALRT'）资源	6-150
图 6-44	编译后的项目列表（‘DITL’）资源	6-152
图 6-45	编译了的按钮、复选框、选项按钮、静态文本项目，或可编辑文字项目的结构	6-153
图 6-46	编译后的控制部件、图标和图片项目的结构	6-154
图 6-47	编译后的定义应用程序的项目的结构	6-155
图 6-48	编译了帮助项目的结构	6-155
图 6-49	编译的项目彩色表资源的结构	6-159
图 6-50	编译了的控制部件色彩表的结构	6-161
图 6-51	编译了的文本字形表的结构	6-162
表 6-1	对话管理器例程的新名称和以前的名称之间的映像	6-102
程序清单 6-1	一个警告资源的 Rez 输入	6-19
程序清单 6-2	根据警告阶段来指定不同的警告响应	6-21
程序清单 6-3	创建您自己的示警声过程	6-22
程序清单 6-4	一个对话资源的 Rez 输入	6-24
程序清单 6-5	用于为警告栏提供项目的 Rez 输入	6-27
程序清单 6-6	使显示矩形的间隔一致的 Rez 输入	6-34
程序清单 6-7	用于定义一个对话框（其中包含有一个弹出式菜单）的对话资源及其项目列表资源的 Rez 输入	6-43
程序清单 6-8	用于一个弹出式菜单的控制资源和菜单资源的 Rez 输入	6-43
程序清单 6-9	使用 ParamText 过程来替代文字字符串	6-47

程序清单 6-10	指定 ParamText 应该在哪里替代一个警告栏消息中的文字	6-48
程序清单 6-11	在一个项目列表中指定可编辑文字项目	6-49
程序清单 6-12	在一个可编辑文字项目中获取由用户键入的文字	6-49
程序清单 6-13	给一个已存在的对话框添加项目	6-54
程序清单 6-14	用于一个对话框及其增补项目的 Rez 输入	6-55
程序清单 6-15	用于在一个项目列表中的定义应用程序的项目的 Rez 输入	6-57
程序清单 6-16	安装用于一个定义应用程序的项目绘制过程	6-58
程序清单 6-17	创建一个可以在系统设定按钮的周围绘制黑体轮廓线的绘制过程	6-59
程序清单 6-18	在显示警告栏之前停用前面的窗口	6-65
程序清单 6-19	使用 GetAlertStage 来确定何时停用前面窗口	6-66
程序清单 6-20	确保创建不定模式对话框之前该对话框还没有打开	6-67
程序清单 6-21	调整各种窗口中的菜单	6-70
程序清单 6-22	针对一个含有可编辑文字项目的定模式对话框，使菜单无效	6-70
程序清单 6-23	为定模式对话框调整编辑菜单	6-72
程序清单 6-24	使用系统系统设定色彩的对话彩色资源的 Rez 输入	6-75
程序清单 6-25	在空事件期间使用 DialogSelect	6-79
程序清单 6-26	响应定模式对话框中的事件	6-83
程序清单 6-27	一个典型的用于警告和定模式对话框的事件过滤函数	6-88
程序清单 6-28	处理所有窗口的鼠标按下事件	6-91
程序清单 6-29	使用 DialogSelect 函数响应鼠标按下事件	6-92
程序清单 6-30	响应关闭命令隐藏不定模式对话框	6-94
程序清单 6-31	核查涉及 Command 键的键按下事件	6-95
程序清单 6-32	核查不定模式对话框中的键按下事件	6-96
程序清单 6-33	响应不定模式对话框中的键按下事件	6-96
程序清单 6-34	激活不定模式对话框	6-98
程序清单 6-35	更新不定模式对话框	6-99

第七章 Finder **界面** 7-1

图 7-1	桌面环境的一个窗口中的应用程序和文稿图标	7-4
图 7-2	为一个应用程序图标自定的旁述	7-5
图 7-3	确认一个应用程序不存在的 Finder 消息	7-5
图 7-4	一个公司的产品系列的大黑白应用程序图标	7-12
图 7-5	系统设定大黑白图标	7-12
图 7-6	一个应用程序的黑白图标和它的掩码	7-13
图 7-7	图符的 ResEdit 模式	7-14
图 7-8	在束资源中链接图标列表资源和文件参考资源	7-23
图 7-9	应用程序不存在时的系统设定警告栏	7-27

图 7-10	应用程序不存在时指定应用程序名称的警告栏	7-29
图 7-11	具有自定消息的应用程序不存在警告栏	7-30
图 7-12	对于 'TEXT' 和 'PICT' 类型文稿的应用程序不存在警告栏	7-30
图 7-13	信息窗口中的版本数据	7-33
图 7-14	应用程序图标的系统设定和自定旁述	7-38
图 7-15	系统文件夹和有关的文件夹	7-42
图 7-16	作为字符串（'STR'）资源编译的署名资源的结构	7-57
图 7-17	编译了的图标列表（'ICN#'）资源的结构	7-58
图 7-18	编译了的小图标列表（'ics#'）资源的结构	7-59
图 7-19	编译了的大 4 位彩色图标列表（'icl4'）资源的结构	7-60
图 7-20	编译了的小 4 位彩色图标列表（'ics4'）资源的结构	7-61
图 7-21	编译了的大 8 位彩色图标列表（'icl8'）资源的结构	7-62
图 7-22	编译了的小 8 位彩色图标列表（'ics8'）资源的结构	7-63
图 7-23	编译了的图标（'ICON'）资源的结构	7-64
图 7-24	编译了的文件参考（'FREF'）资源的结构	7-65
图 7-25	编译了的束（'BNDL'）资源的结构	7-66
图 7-26	在一个束资源中把局部标识符映射到图标列表资源标识符	7-67
图 7-27	束资源中文件参考资源多余的局部标识符映射的结构	7-67
图 7-28	编译了的未找到应用程序名字字符串资源的结构	7-68
图 7-29	编译了的未找到应用程序信息字符串资源的结构	7-69
图 7-30	编译了的版本（'vers'）资源	7-70

程序清单 7-1	Rez 输入格式的一个署名资源	7-8
程序清单 7-2	一个图标列表资源的 Rez 输入格式	7-14
程序清单 7-3	Rez 输入格式的文件参考资源	7-19
程序清单 7-4	Rez 输入格式的束资源	7-21
程序清单 7-5	未找到应用程序名字字符串资源的 Rez 输入格式	7-28
程序清单 7-6	在文稿的资源分支中储存一个未找到应用程序名字字符串资源	7-28
程序清单 7-7	把未找到应用程序名字字符串资源拷贝到文稿的资源分支	7-29
程序清单 7-8	未找到应用程序信息字符串资源的 Rez 输入格式	7-30
程序清单 7-9	一对版本资源的 Rez 输入格式	7-32
程序清单 7-10	大小资源的 Rez 输入格式	7-35
程序清单 7-11	确定一个文稿是否固定样板文稿	7-36
程序清单 7-12	应用程序图标的旁述资源的 Rez 输入格式	7-39
程序清单 7-13	使用 ResolveAliasFile 函数打开文件	7-41

前　　言

关于本书

本书名为《Inside Macintosh 工具箱基础》(*Inside Macintosh: Macintosh Toolbox Essential*)，它介绍了 Macintosh 应用程序实质性的部分以及您可以使用执行的系统软件例程。

如果您是 Macintosh 电脑编程方面的新手，您还需要阅读《Inside Macintosh 概述》(*Inside Macintosh: Overview*) 一书，那本书介绍了 Macintosh 编程的一般概念和《Macintosh用户界面设计原则》(*Macintosh Human Interface Guidelines*)，它对每个 Macintosh 应用程序应该遵循的用户界面准则和原理均进行了全面而详尽的讨论。

本书不仅介绍了事件、窗口、菜单、控制、警告栏和对话框，而且还讨论了您的应用程序应如何与 Finder 交互作用等问题。

Macintosh 应用程序，能够对用户动作以及其它与硬件和软件相关的事件做出响应。如果希望您的应用程序对事件（例如键盘输入、鼠标输入、屏幕窗口外观的更改以及您的应用程序处理状态的更改等）做出响应，请参阅本书的“事件管理器”一章。

若要创建菜单并设置您的应用程序的菜单栏，请参阅“菜单管理器”一章。这一章介绍了如何在您的菜单中定义项目，如何启用及停用菜单，如何使用户选择菜单命令项目，以及在用户选择一个菜单命令项目后如何响应。

若要创建用户在其中显示或编辑信息的窗口，请参阅“窗口管理器”一章。这一章介绍了窗口的基本类型，并讨论了您的应用程序如何才能与窗口管理器一起工作，以支持与窗口操纵（例如移动窗口、缩放窗口及调整窗口大小）有关的标准用户界面规则。

若要在应用程序窗口中创建控制部件（例如滚动杆的控制）或在对话框中创建控制部件（例如按钮或复选框的控制），请参阅“控制管理器”一章。

若要创建对话框或警告栏这类您的应用程序用来和用户通信或从用户那里寻求信息的窗口，请参阅“对话管理器”一章。

若要为您的应用程序及其建立的文稿创建图标，请参阅“Finder界面”一章。这一章还介绍了文件类型和创建者，并介绍了Finder为显示您的应用程序及其创建的文稿所需要的各类资源（图标、文件参考资料和束）。

在了解了本书介绍的 Macintosh 应用程序的基本部分后，您就可以添加额外的功能，例如像《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中介绍的旁述和对拷贝及贴的支持等功能。您还可以在《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中找到有关资源管理器的详细信息。

前　言

了解了如何创建菜单、窗口和对话框后，您就能够通过将数据写入一个文件，来存储用户输入的信息。您还能够打开原来存储的文件并将其中的信息读入窗口。您可以使用文件管理器来打开、读、编写及关闭文件。请参阅《Inside Macintosh 文件》(*Inside Macintosh : File*) 一书“文件管理介绍”一章中关于如何读写文件的信息。

请参阅《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*) 一书中关于在窗口中和其它图形端口绘图的信息。

请参阅《Inside Macintosh 文本》(*Inside Macintosh : Text*) 一书中关于处理应用程序中的文字的信息。

请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh : Interapplication Communication*) 一书中关于和其它应用程序通讯的信息。

典型章节的格式

本书中几乎所有的章节都遵循一种标准的结构。例如，“事件管理器”一章包含以下几节：

- “事件介绍”。本节对您的应用程序可以接收的事件类型作了一般性的介绍
- “关于事件管理器”。本节对事件管理器提供的功能作了综述
- “使用事件管理器”。本节介绍了您可以用事件管理器完成的任务。它阐述了如何使用最常用的例程，给出了有关的用户界面信息，提供了代码段示例，并补充了一些信息
- “事件管理器参考资料”。本节通过说明数据结构、例程及其使用的资源，对事件管理器作了全面的介绍。每个例程的说明也遵循一个标准的格式，首先是例程说明，然后是例程每个参数的说明。一些例程的说明还给出了一些额外的说明信息，例如汇编语言信息或结果代码
- “事件管理器小结”。本节提供了与事件管理器相关的常数、数据结构、例程和结果代码的 Pascal 和 C 界面。它还包括了有关的汇编语言界面信息

本书中使用的一些惯例和规则

《Inside Macintosh 工具箱基础》(*Inside Macintosh : Macintosh Toolbox Essentials*)一书在编写过程中使用了多种惯例规则。需要专门处理的词使用了特殊的字体和字形。为了使您能够迅速浏览书中的信息，一些信息(例如寄存器的内容）还使用了专门的格式。

前　言

特殊字体

所有的代码段列表、保留词和真实数据结构、域、常数、参数及例程的名称均采用Courier字体书写（这是 Courier）。

以黑体形式出现的词是在词汇表中定义的关键术语或概念。

附注的类型

本书中使用了几种类型的附注。

附注

这种类型的附注包含的是有趣的但对理解正文不一定是必要的信息。（在 2-7 页中有这样一个实例）

重要告诫

这种附注包含着对理解正文必不可少的信息。（在 5-27 页有这样一个实例）

警告

这样的警告表示在您设计应用程序时必须注意的潜在问题。不注意这个警告可能会导致系统崩溃或数据丢失。（在 2-105 页中有这样的一个实例）

空字符串

本书有时会要您在例程参数和资源中提供一个空字符串。如何指定一个空字符串取决于您使用的语言和开发环境。例如，在 Rez 输入文件和 C 代码中，您可以用两个双引号（“ ”），在 Pascal 中用两个单引号（‘ ’）来指定空字符串。

汇编语言信息

《Inside Macintosh》提供了如下有关特殊例程寄存器的信息：

寄存器在输入时

A0　　输入时寄存器 A0 的内容

寄存器在输出时

D0　　输出时寄存器 D0 的内容

在每一章结尾的“汇编语言小结”一节中，《Inside Macintosh工具箱基础》（*Inside Macintosh : Macintosh Toolbox Essentials*）以下述格式给出了数据结构域的信息：

0	what	字	事件代码
2	message	长型	事件消息
6	when	长型	启动后过去的时间 Tick

左边一列表明了从数据结构开始，域字节的偏移；第二列显示了域的名称，该名称与 MPW ASCAL 界面文件中定义的名称一样；第三列表明该域的大小；第四列对域的使用提供了简要的说明。若要了解每一域的详尽说明，请参阅该章参考一节中有关数据结构的讨论。

开发环境

本书中所介绍的系统软件例程在使用 Pascal、C 或汇编语言界面时可以使用。您如何存取这些例程取决于您正在使用的开发环境。本书使用 Macintosh Programmer's Workshop（MPW）的 Pascal 界面来显示系统软件例程。

本书中列出的所有代码段均采用 Pascal 表示（说明资源的列表除外，它用 Rez 输入格式列示）。它们介绍了使用各种例程的方法，并使用图解说明了完成特定任务的技术。所有的代码段列表都已经编译过，并在大多数情况下测试过。但是，苹果电脑公司并不希望您在应用程序中使用这些代码段示例。您可以在图形、表格和列表中找到代码段列表的位置。如果您知道代码段列表中特定例程的名称（例如 DoEvent 或 MyAdjustMenu），您就能够在本书索引中的“例程示例”中找到例程所在的页码。

为了使本书中的代码段列表更容易阅读，本书仅列示了有限的错误处理内容。您需要开发自己的错误处理技术。

本书有时参考一个称为 SurfWriter 的示例应用程序来说明概念，这不是苹果电脑公司的一个真正产品。

APDA 是全世界有关苹果公司各种信息的汇总，包括 300 多种开发工具，技术资源、培训产品及任何希望在苹果平台上开发应用程序的人员感兴趣的信息。用户收到的 APDA 工具目录簿季刊中列有所有苹果公司和最常用的第三方开发工具的最新版本。它的订阅非常简单，无需支付成员费。大多数产品不需要申请表。APDA 提供了方便的付款和交付方式选择，包括现场授予许可证。

前　言

如果希望定购产品或索取 APDA 工具目录簿，请和以下地址联系：

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

电话：	800-282-2732（美国）
	800-637-0029（加拿大）
	716-871-6555（世界其他各地）
传真：	716-871-6511

如果您提供的是商用产品和服务，需要了解有关苹果公司开发商支持计划的信息，请拨电话 408-974-4897。

有关注册署名、文件类型、苹果事件和其它技术信息，请和以下地址联系：

Macintosh 开发商技术支持中心
苹果电脑公司
20525 Mariani Avenue, M/S 75-3T
Cupertino, CA 95014-6299

第一章　Macintosh 工具箱引论

Macintosh 工具箱概述	1-4
事件	1-5
菜单	1-6
窗口	1-6
控制	1-7
警告栏和对话框	1-8
图符及其它和 Finder 的交互作用	1-10
资源	1-11
旁述	1-14
拷贝和粘贴	1-14
相关的系统软件功能	1-14
在屏幕上绘图	1-14
处理文字	1-14
管理文件	1-15
分配存储器和启用进程	1-15
创建版本和引用程序	1-15
和其它应用程序通讯	1-16
设计您的应用程序	1-16

本章是有关Macintosh工具箱提供的功能的引论。Macintosh 工具箱是系统软件例程的集合，它能够使您的应用程序保持一致和标准的界面，这些例程还能够简化您的应用程序可能需要执行的其它任务。

典型的 Macintosh 应用程序可以为用户提供友好、直观、易于使用且可视的界面。精心设计的 Macintosh 应用程可以使用户自由地根据他们的需要执行动作，完成任务。这种精心设计思想的目的就是要让用户更有效、更轻松地使用他们的电脑。一般来说 Macintosh 应用程序的用户应该始终能够自由地选择他或她下一步要执行的动作。(这是事件循环的基本宗旨，在本书的“事件管理器”一章中有更详细的叙述)。

图 1-1 列出的是用户和典型的 Macintosh 应用程序（例如 SurfWriter ）对话时可能出现的屏幕。SurfWriter 应用程序是一个用户用来进行简单的文字编辑的应用程序。和大多数 Macintosh 应用程序一样，SurfWriter 使用了以下几种用户界面元素:

- 用户选择命令的菜单
- 允许用户输入和编辑信息的窗口
- 允许用户在窗口中显示更多信息的滚动杆

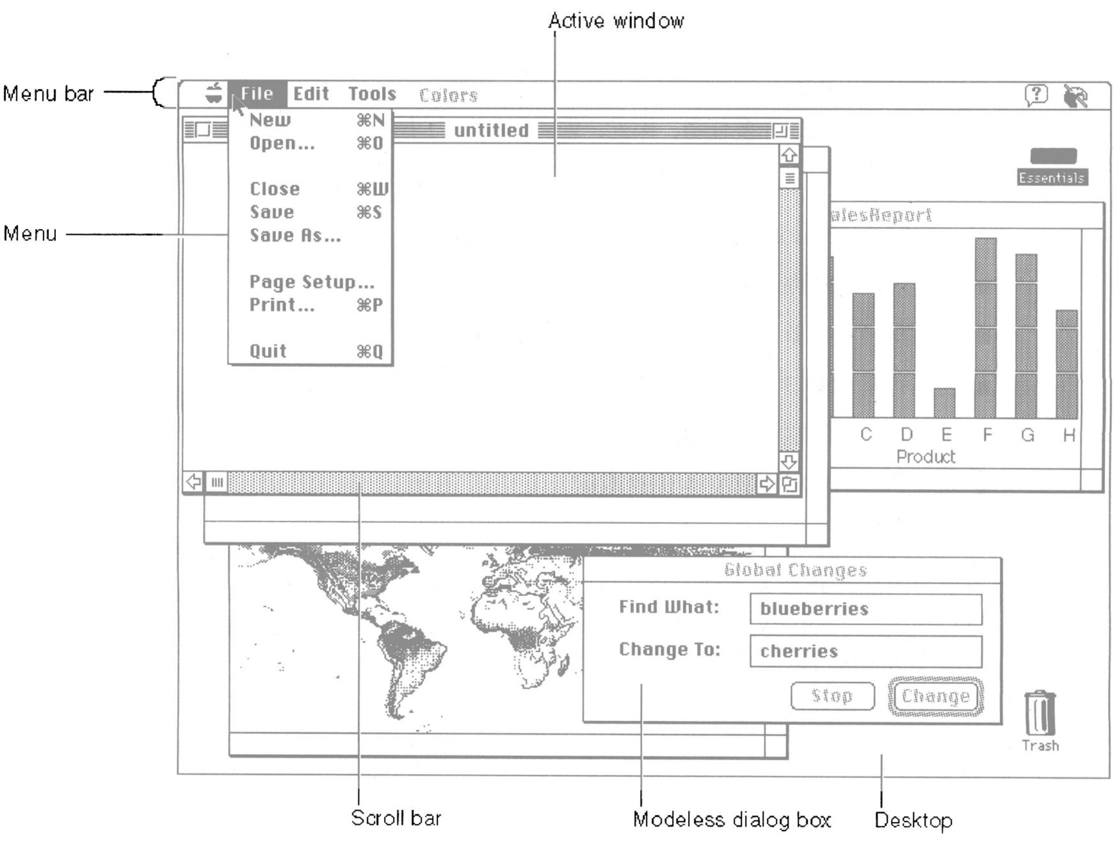


图 1-1 在桌面环境下拥有多个窗口的 SurfWriter 应用程序

■ 允许用户控制各种设定状态或选项的其它控制部件（例如更改按钮）
■ 向用户征求信息的对话框

您可以创建一个应用程序将这些用户界面元素结合进去，并充分利用 Macintosh 工具箱所提供的例程帮助用户完成具体任务。

Macintosh工具箱概述

Macintosh 系统软件中包含着一组功能强大的例程，您的应用程序能够使用它们创建窗口、管理菜单、为对象涂色、显示文字、打开文件、在程序间共享数据、打印文件，并执行许多其它有用的任务。

Macintosh 工具箱拥有许多系统软件例程，其中多数例程（并非全部）都有助于向用户提供您的应用程序界面。这些例程中包括了由事件管理器、菜单管理器、窗口管理器、控制管理器、对话管理器、帮助管理器、资源管理器及剪贴数据存储区管理器提供的一些例程。

您可以在您的应用程序中直接调用这些例程。通过使用系统软件例程，您能够利用它们来为您完成许多任务，因此，您可以把精力集中在开发您的特定产品所特有的应用程序部分。

使用 Macintosh 工具箱，您可以执行以下任务：

■ 响应用户的动作，例如鼠标动作或键盘输入
■ 创建及显示菜单
■ 创建和显示窗口、警告栏及对话框
■ 在窗口、警告栏及对话框中创建和显示控制部件
■ 为您的应用程序及其文件创建图标

《Macintosh 工具箱基础》(*Macintosh Toolbox Essential*) 这本书介绍了 Macintosh 应用程序的这些基本元素。《Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书介绍了 Macintosh 应用程序的一些附加功能，您可以使用这些功能执行以下任务：

■ 为您的应用程序的菜单、窗口和对话框创建旁述
■ 支持拷贝和贴
■ 设定您的应用程序所需的菜单、窗口、控制、对话框以及旁述等资源的特征，以便调用

最好的 Macintosh 应用程序，是根据《Macintosh 用户界面设计原则》(*Macintosh human Guidelines*) 一书中的准则设计的。您设计的应用程序应始终以满足用户的需要，并以一致和预期的方式响应用户动作为宗旨。

《Macintosh 用户界面设计原则》(*Macintosh human Guidelines*) 介绍了：

- Macintosh 界面的哲学和设计原则
- Macintosh 界面的各部分，包括界面组成部分和使用
- 为 Macintosh 产品设计人机界面的方法

通过用户测试，您便经常能够获得对您的应用程序设计有价值的反馈。您需要尽早进行应用程序的可使用性测试，通常应在您的应用程序的开发阶段进行。

事件

应用程序的事件循环是每个 Macintosh 应用程序的核心。事件循环是指应用程序中对用户动作进行处理和响应的那段代码。您可以使用事件管理器来检索关于这些动作的信息。您可以得到各种信息，例如，告诉您的应用程序用户是按了键盘还是按了鼠标器按钮；由于用户移动了窗口，您的应用程序的一个窗口是否需要更新；或者您的应用程序是否需要对某些其它硬件或软件的动作做出响应。

您的应用程序的结构设计应使您的应用程序能对事件做出响应，使用户能够按任意次序执行任务。例如，用户应该能够在窗口中键入文字，选取一个图形并复制它，打开一个新的文件，在其中贴图形，打开一个新文稿，然后回到第一个窗口选定文字，更改其字体、大小或字形。

您的应用程序对事件响应的方式应是：允许用户根据需要随时在您的应用程序和其他应用程序间切换（例如通过点按属于另一应用程序的窗口）。您的应用程序还应该在其不占用CPU处理器时间时为其它应用程序让出时间。系统软件提供了一个合作的多任务环境，它使得用户能够在许多已打开的应用程序间切换，这使应用程序在其它应用程序不占用处理器时，能有可用的处理时间。系统软件在您的应用程序和其它应用程序间协调进程时间的分配。

为了从其它应用程序征求服务或信息，或者向其它应用程序提供服务，您还可以让您的应用程序和其它应用程序进行通讯。为此，您可以使用事件管理器或苹果事件管理器。

本书中的“事件管理器”一章讲述了构造您的事件循环和事件处理代码段的方法，从而为接收有关用户动作和您的应用程序处理状态改变的信息提供了保障，同时还讲述了如何与其它应用程序通讯以及如何对这些事件做出响应等内容。

菜单

菜单是 Macintosh 应用程序设计的重要部分。菜单允许用户从您的应用程序提供的选定项和命令列表中显示或选择一个项目。您可以根据您的应用程序的动作或任务设计您的菜单。所有的应用程序都应该支持苹果、文件、编辑、帮助、键盘和应用程序等六个标准菜单。1-3 页中的图 1-1 显示了SurfWriter 应用程序的文件菜单。系统软件可令您十分方便地创建下拉式菜单、嵌套式菜单和弹出式菜单。本书的“菜单管理器”一章叙述了如何创建您的应用程序菜单、如何设置您的菜单栏、如何显示菜单并对用户从菜单中获取一个菜单命令项目做出响应等内容。

窗口

大多数应用程序都可以通过窗口和用户对话。图 1-2 是一个普通窗口及其组成部分。本书的“窗口管理器”一章介绍了您的应用程序可以创建的窗口类型以及如何对用户涉及窗口的动作做出响应等内容。

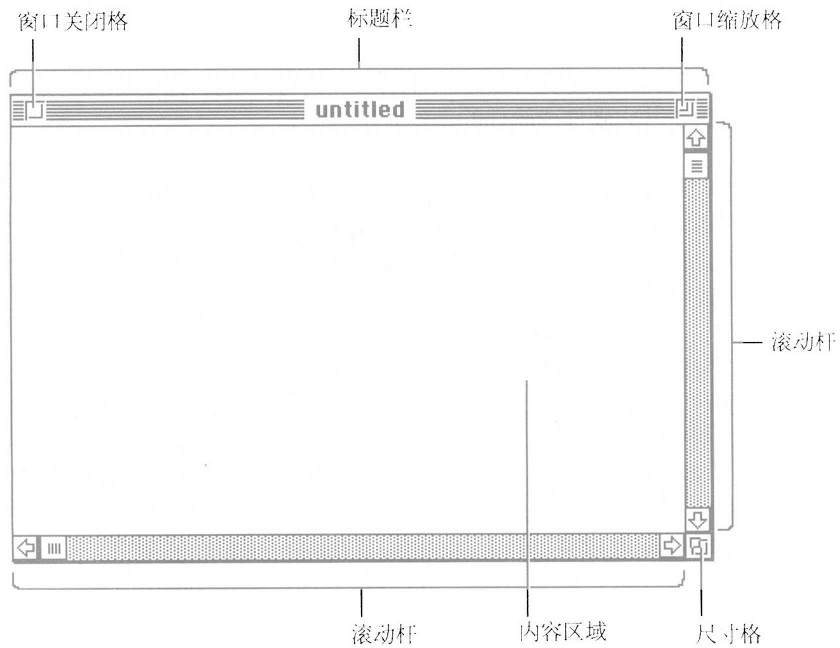


图 1-2 一个典型的窗口

用户一般在桌面环境中有一个或多个窗口，它们通常来自不同的应用程序。尽管用户可以在桌面环境中有多个窗口，但只有一个窗口是当前活动窗口。当前活动窗口是在桌面环境中出现在最上面的窗口，并可以用其标题栏中的跑道线来识别。图 1-2 显示了一个当前活动窗口；在 1-3 页的图 1-1 中，标题为 SalesReport 的窗口是一个非当前活动窗口。

所有的键盘活动都是直接针对当前活动窗口的。您应该确保您的应用程序遵循人机界面有关当前活动窗口和非当前活动窗口的准则。例如，您应该显示滚动杆，并醒目显示属于您的应用程序的现有窗口中的任何选项；而且您还应该隐藏属于您的应用程序的非当前活动窗口中的滚动杆，并取消其中醒目显示的任何选定项。您的应用程序的菜单栏还应始终反映您的应用程序当前活动窗口的状态—即您的应用程序应该只允许属于当前活动窗口的那些菜单命令有效。

当您的应用程序窗口需要创建、移动、调整大小、缩放或更新窗口内容时，您可以使用系统软件例程来帮助。本书中“窗口管理器”一章会告诉您应该如何完成这些任务。

控制部件

大多数窗口和对话框均含有控制部件。控制部件是一种用户可以用鼠标操纵的屏幕对象，利用它可以作出直接影响您的应用程序的动作，或是为了修正将来的动作而改变设定状态。

按钮、复选框、选项按钮、弹出式菜单和滚动杆是大多数应用程序所使用的一般控制部件的实例。复选框、选项按钮和弹出式菜单常用在对话框中；按钮在警告栏或对话框中用得最多；滚动杆在窗口中用得最多。图 1-3 列示的就是这些类型的控制部件。

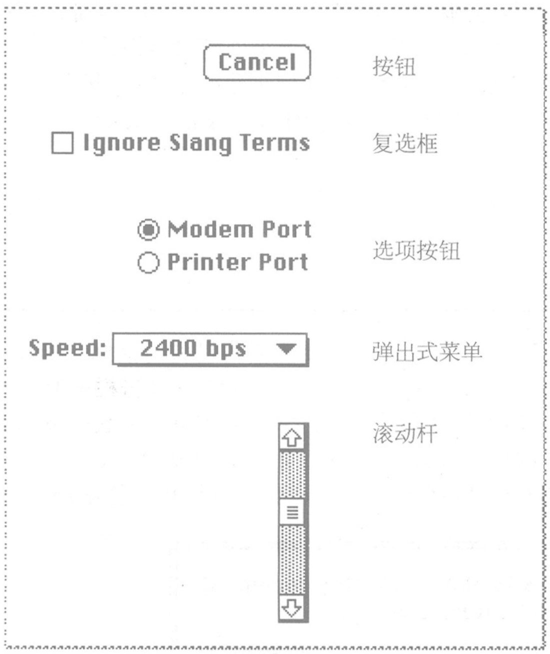


图 1-3　一般控制部件

按钮的外观是一个圆角的矩形框，其标题在矩形框中央。当用户完成了对话框定义的操作或接受了警告栏中的错误消息后，可以点按按钮来执行一个确认或取消的结束动作。

复选框是一个小方框，其标题在方框的旁边。当复选框的有关设定状态为有效时，小方框内有一个 选择标记“×”，无效时框内是空白。复选框指示的选项只能是有效或无效。

选项按钮是一个圆圈，其标题在圆圈的旁边。当选项按钮的有关设定状态有效时，圆圈内有个小黑点，无效时中间是空白。选项按钮和复选框相似之处在于它们均保持并显示有效和无效设定状态；但是，一组选项按钮中同时只能有一个处于有效状态。您必须决定如何将您的选项按钮进行分组，您的应用程序必须保证一组选项按钮中只有一个处于有效状态。

弹出式菜单是出现在对话框或窗口中的菜单。您可以用弹出式菜单代替选项按钮，使用户在一系列选项或设定状态中进行选择。

滚动杆的外观是一个淡灰色的矩形长杆，在矩形长杆的两端都有一个滚动箭头。窗口可以有水平滚动杆、垂直滚动杆或兼而有之。您可以使用滚动杆让用户改变窗口中可显示的文稿部分。

您可以使用控制管理器的例程来跟踪和响应用户的控制、屏幕刷新控制和操纵控制等动作。通常情况下，您可以使用对话管理器来处理对话框或警告栏中的大多数控制部件。本书中“控制管理器”一章介绍了如何创建控制（特别强调在窗口中创建及使用滚动杆）的方法，“对话管理器”一章则提供了更多的有关如何在对话框和警告栏中创建控制部件的信息。

警告栏和对话框

除了标准窗口外，您的应用程序还可以使用警告栏和对话框。警告栏是您的应用程序向用户发出警告或向用户报告错误信息的一个窗口。警告栏一般由说明性的文字和用户用来接受或更正问题的按钮组成。图 1-4 是 SurfWriter 应用程序在用户未存储文稿就要关闭一个窗口时显示的一个警告栏。该警告栏为用户提供了一个在 SurfWriter 应用程序关闭窗口前存储文稿的机会，这样可以使用户避免丢失数据。

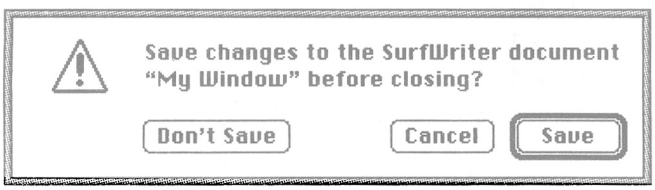


图 1-4　一个警告栏

对话框是一个可以用来请求用户输入附加信息的窗口。对话管理器可以帮助您显示对话框并能够提供与用户对话的标准方法。对话框可以包含可编辑的文字项目、提供信息或指示性的文字以及象按钮和复选框这样的控制部件。您可以创建定模式对话框、可移动定模式对话框或不定模式对话框。图 1-5 为上述每类对话框各显示了一个实例。

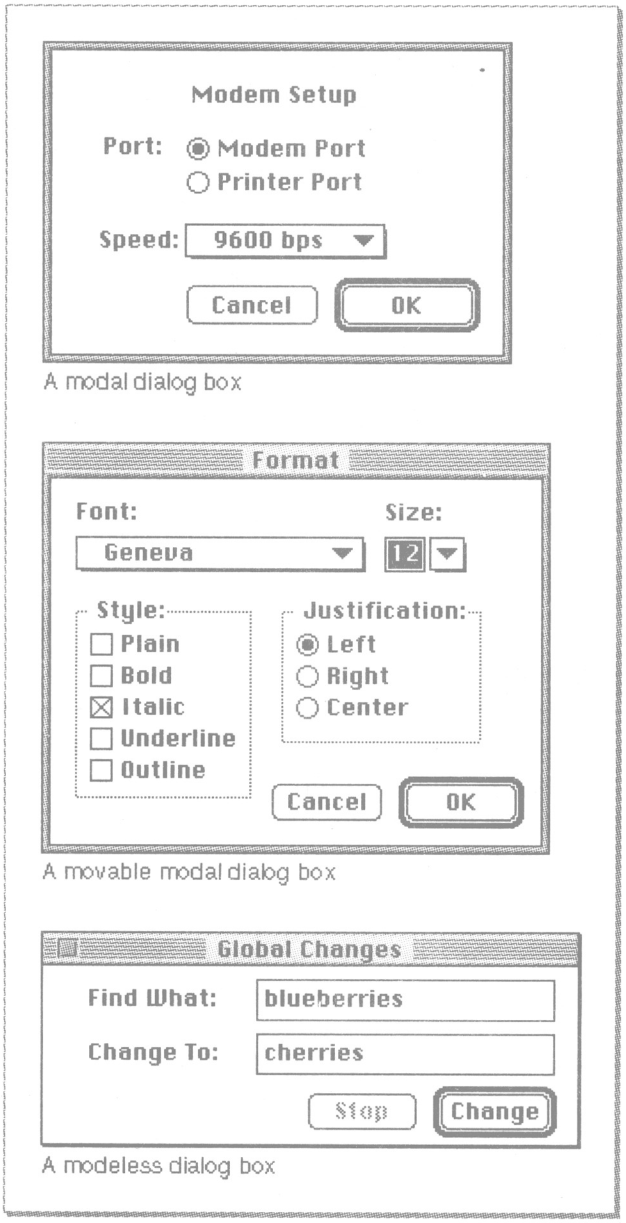


图 1-5　定模式对话框、可移动定模式对话框及不定模式对话框

定模式对话框将用户置于仅能在对话框内工作的状态或“定模式”。定模式对话框除了可以含有可编辑的文字项目及附加控制部件(例如选项按钮和弹出式菜单)外，它在外观上和警告栏类似。用户不能移动一个定模式对话框，只能通过点按其按钮取消它。只有当用户必须在执行其它工作前完成某个操作时，您才可以使用定模式对话框。

可移动定模式对话框是带有标题栏（但没有关闭格）的定模式对话框，用户可以使用标题栏移动对话框。用户可以通过点按对话框按钮来取消对话框；但是，当您使用可移动定模式对话框时，您应该允许用户通过点按另一个应用程序窗口或者从苹果菜单或应用程序菜单上选择另一个应用程序来实现应用程序切换。当用户可能需要移动对话框来显示屏幕的其它部分时，或者用户能够切换到别的应用程序而不影响您的应用程序的状态时，请使用一个可移动定模式对话框。

不定模式对话框是一个看上去象没有尺寸格或滚动杆的文稿窗口的对话框。不定模式对话框并不要求用户在做其它事情前做出响应。用户可以移动一个不定模式对话框，在不定模式对话框和其它窗口间切换，并像文稿窗口一样关闭不定模式对话框。只要可能，您应该使用不定模式对话框来代替定模式对话框或可移动定模式对话框。为使用户能够执行其它操作（例如在文稿窗口中工作）而不取消不定模式对话框时，请使用不定模式对话框。

本书中“对话管理器”一章详细介绍了您怎样才能够为应用程序创建警告栏和对话框。

图标及其它和 Finder 的交互作用

一旦设计了应用程序，您就必需为该应用程序和它所创建的文稿创建相应的图标。Finder会为用户显示这些图标。如果您的应用程序在苹果菜单或应用程序菜单中作为一个项目出现，菜单管理器将在其名称的旁边显示您的应用程序图标，当您的应用程序成为现用应用程序时，菜单管理器会将您的应用程序图标作为应用程序菜单的当前标题。

本书中“Finder 界面”一章介绍了为您的应用程序及其文稿创建图标的方法。同时还介绍了您的应用程序如何与 Finder 交互作用等内容。

当一个用户打开您的应用程序或者打开或打印它创建的一个文稿时，Finder使用进程管理器为您的应用程序安排执行时间，然后设置您的应用程序需要的信息以便确定（如果需要）应打开或打印的文稿。在 System 7 中，您的应用程序可以选择通过苹果事件来接收这一信息。通过支持这些及其它苹果事件，您的应用程序定能有效地响应用户和其它应用程序的请求。请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplicaton Communication*) 一书，关于支持苹果事件的相关信息。

资　源

资源是每个Macintosh应用程序组成的基本元素。通过在资源中定义菜单、窗口、控制、对话框、语音、字体和图符的说明，您可以大大简化这些元素和其它元素的创建和管理。使用资源还可以使用户界面元素非常容易的转换成其它语言，即大大减少了程序的本地化工作量。

资源是根据定义的结构存放在文件资源分支中的所有数据 资源中的数据是根据其资源类型来解释的。您通常使用资源编译程序或资源编辑程序来创建资源。本书采用 Rez 格式显示资源; Rez是Macintosh Programmer Workshop提供的一种资源编辑程序（可从APDA 中得到)。您也可以使用其它资源工具，例如ResEdit（也可从APDA中得到）来创建您的应用程序资源。

本书介绍的大多数管理器都是使用资源管理器来为您读取资源的。例如，您可以使用菜单管理器、窗口管理器、对话管理器及控制管理器从资源中读取您的应用程序菜单、窗口、对话框及控制部件的说明。一旦一个资源数据被读入存储器，所有这些管理器都会相应地解释这些数据。一般情况下，您不仅可以使用这些管理器来存取资源，而且还能直接使用资源管理器来读写资源。

《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中“资源管理器”一章详细介绍了资源管理器。但是，为了帮助您更好地了解菜单管理器、窗口管理器、对话管理器及控制管理器如何使用资源，本节给出了关于资源的简要概述，并对资源管理器进行了一般性的介绍。

Macintosh系统软件将一个文件作为储存在一个Macintosh卷并分成两分支的有名称的、有序的字节序列来处理，这两分支分别称为数据分支和资源分支。数据分支包含用户通常创建的那些数据；应用程序创建文件能够以适当的方式储存并解释数据分支中的数据。文件的资源分支由资源映像和资源本身组成。

当您将数据写入文件时，您可以将其写在文件的资源分支或数据分支中。通常，您使用文件管理器的例程读或写文件的数据分支，使用资源管理器的例程读或写文件的资源分支。一般情况下，您储存的资源数据都是具有确定结构的数据，如图标和语音，以及菜单、控制部件、对话框和窗口的说明等。当您创建一个资源时，您会为它指派一个资源类型和资源标识符。资源类型是一个包含 4 个字符的序列，是一种特定资源类型的唯一识别标志。资源标识符是该资源类型中的一个特定的资源编号(您也可以使用资源名称代替资源标识符，从而可以在一个资源类型中识别一个特定的资源)。例如，在一个资源中创建一个菜单资源，您必须指派该资源类型为 'MENU'，并给它定一个与其它已定义的 'MENU'资源不同的资源标识符和资源名称。有些资源对能用作资源标识符的数字有一定限制一般来说，您可以使用 128 至 32767 之间的数字。

系统软件定义了许多标准的资源类型，例如 'ALRT'、'CNTL'、'CODE'、'DITL'、'DLOG'、'FONT'、'ICN#'、'ICON'、'MBAR'、'MENU'、'STR'、'STR#' 及 'WIND'。您可以使用这些资源类型来定义它们相应的元素（例如使用 'WIND' 资源来定义一个窗口）。如果您的应用程序需要建立不同于系统软件定义的标准资源类型，您也可以创建您自己的资源类型。

资源管理器不对单个资源类型的格式进行解释。当您要求一个带有给定资源标识符 的特定类型资源时，资源管理器会在指定资源中寻找，如果找到，便将该资源读入存储器并给它回送一个间接指针。您的应用程序或其它系统软件例程可以使用资源管理器将资源读入存储器。例如，当您使用窗口管理器从 'WIND' 资源中读一个窗口的说明时，窗口管理器使用资源管理器将该资源读入存储器。资源进入存储器后，窗口管理器对资源的数据进行解释，并根据资源说明的特征创建一个窗口。

系统软件在系统文件中储存某些自己使用的资源。虽然这些资源中的多数只有系统软件才使用，如果需要，您的应用程序也可以存取其中的一些资源。例如，I 型光标和手表光标的标准影像可以作为资源类型 'CURS' 储存在系统文件中。同时，您还可以使用这些资源来更改应用程序光标的外观。

有时，您需要将资源写入文件的资源分支中。例如，如果您的应用程序存储了窗口的上次位置及窗口的大小（根据用户的决定），您可以将此信息储存在文稿资源分支中的一个您的应用程序定义的资源中。用户下次打开文稿时，您的应用程序能够读出存储在该资源中的位置并相应地为文稿定位。

通常您是在您的应用程序的资源分支中储存该应用程序的专用资源的，例如应用程序菜单、窗口、控制部件和对话框的说明。您可以将专为您的应用程序创建的文稿资源储存在文稿文件的资源分支中。

一个文件资源分支中的资源映像包含多个入口地址，这些入口地址可以在资源分支中为每个资源提供一个位置。当资源管理器打开一个文件的资源分支时，它便将资源映象读入存储器。资源管理器在将资源读入存储器时，它会用存储器中数据的间接指针代替资源映像中的入口地址。资源管理器搜索一个资源时，总是在搜寻存储器中的资源映像，而不是搜索磁盘资源分支中的资源映像。如果要求的资源在存储器中，资源管理器就使用存储器中的资源。不然的话，它会从磁盘资源分支的资源映像中将资源读入存储器。

资源管理器打开一个资源分支并将其资源映像读入存储器后，它将资源映像一直保留在存储器中，直至文件关闭。您可以指定：在资源管理器打开文件资源分支时，立即将一个资源读入存储器，也可以指定：仅在需要时，资源管理器才将资源读入存储器。

资源管理器可以将您的应用程序打开的资源分支中的资源储存在您的应用程序堆中的可重定位块内。为了给其它数据腾让空间，您也可以指定资源管理器是否应该从存储器中清除一个资源。若您指定的资源是可清除的，在通过其资源间接指针存取它之前，您需要使用资源管理器来确认它是否在存储器中的。

当用户打开您的应用程序时，系统软件会打开您的应用程序的资源分支。当您的应用程序打开一个文件时，它通常将文件的数据分支和资源分支都打开。当您的应用程序向资源管理器请求一个资源时，资源管理器按照一个专门的搜索次序来进行搜索（若需要，您的应用程序可以用资源管理器例程来改变搜索次序）。通常，资源管理器首先在您的应用程序打开的上一个文件的资源分支中搜索资源。所以，当您的应用程序只打开了一个文件时，资源管理器首先搜索的就是该文件的资源分支。如果资源管理器在那里没有找到资源，它会以和您的应用程序打开文件相反的次序搜索它们的资源分支。如果仍未找到资源，就再到系统文件的资源分支中搜索。

这个搜寻路径允许您的应用程序使用系统文件中定义的资源；超越系统文件中定义的资源；通过使用储存在您的应用程序资源分支中的资源实现文件间的资源共享以及超越您的应用程序定义的资源而使用文稿专用资源。

尽管Macintosh文件的资源分支和数据分支之一可能是空的，但它仍然总是同时具有这两种分支。文稿文件通常在数据分支中放置文稿数据，任何文稿专用资源如首选设定状态、窗口位置及文稿图标均放在资源分支中。应用程序的资源分支通常包括说明应用程序菜单、窗口、控制部件、对话框和图符的资源，以及代码段本身。

您是将数据储存在文稿文件的数据分支还是资源分支，在很大程度上取决于您是否能够以有用的方式将数据结构化成一个资源。例如，将文稿专用设定状态（例如文稿的上一次窗口大小和位置）作为资源储存在文稿的资源分支中常常是很方便的。用户可能编辑的数据通常储存在文稿的数据分支中。

一个资源分支最多可以包含2,700个资源。在搜索资源分支的资源类型和资源标识符时，资源管理器采用的是一种线性搜寻方式。一般来说，在任何一个资源分支中，您都不应该创建 500 个以上同样类型的资源。

《Inside Macintosh 工具箱扩展》（*Inside Macintosh: More Macintosh Toolbox*）一书详细介绍了资源和资源管理器的使用。请参阅《Inside Macintosh 文件》（*Inside Macintosh: File*）一书中有关将数据写入文件数据分支的信息。

旁 述

您的应用程序能为窗口菜单、对话框或内容区域这些元素提供旁述。《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中“帮助管理器”一章介绍了您的应用程序怎样才能提供旁述。您也可以使用应用程序 BalloonWriter 为您的应用程序界面的某些元素(例如菜单)创建旁述。BalloonWriter 可以从 APDA 中得到。

拷贝和粘贴

所有的 Macintosh 应用程序都应支持从剪贴板拷贝数据和向剪贴板贴数据这两种功能。《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中的“剪贴数据存储区管理器”一章介绍了如何使用剪贴数据存储区管理器从剪贴板拷贝数据和向剪贴板粘贴数据。

相关的系统软件功能

除了Macintosh工具箱提供的管理器外,您也可以使用其它管理器和系统软件例程。例如,您可以使用 QuickDraw 例程来绘制您的应用程序窗口的内容,使用 TextEdit (和对话管理器一起) 处理对话框中的可编辑文字项目,使用文件管理器来读写文件,使用进程管理器和存储器管理器来控制您的应用程序执行的各个方面,使用编辑管理器和苹果事件管理器来支持应用程序间通讯。本章的余下部分介绍了其中一些管理器,并提示您可以在哪里找到更多的信息。

在屏幕上绘图

系统软件例程 (例如 QuickDraw 所提供的例程) 可以执行所有屏幕上的绘图任务。例如,您的应用程序告诉 QuickDraw 在什么地方画什么,QuickDraw 就会按照应用程序发出的命令在屏幕上进行绘图。系统软件提供的图形例程支持对象的快速绘制,例如圆、椭圆、方框、线、文字和图案等。请参阅《Inside Macintosh 绘图》(*Inside Macintosh: Imaging*) 一书中与图形有关的专门信息。

处理文字

您可以使用 TextEdit 提供的系统软件例程大大简化您的应用程序需要执行的基本文字编辑和格式化工作。例如,大多数应用程序在对话框中使用可编辑文字项目;您的应用程序可以使用对话管理器(它调用 TextEdit)来自动处理可编辑文字项目中与用户的对话。对话管理器和您的应用程序能够使用 TextEdit 来插入新的文字、删

除用户使用退格键扫过的字符、在窗口内滚动文字、剪切文字、复制文字、贴文字、选取文字块以及处理词的换行。大多数应用程序仅在对话框作简单文字编辑时使用TextEdit,在文稿窗口中处理可编辑的文字时则使用它们自己的技术。

您的应用程序应该设计成能处理一种以上的语言或语系的文字。系统软件提供了许多例程来帮助您实现这个目标。例如，当您的应用程序在文稿的底部自动显示日期时，您可以使用 TextUtility 例程使日期显示格式与当前文稿保持一致。类似地，如果您的应用程序提供了一个查找命令，它能够使用 TextUtility 例程根据断词表及当前的文稿来搜索。

请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书有关您怎样在您的应用程序创建的文稿中支持文字编辑的信息，以及有关如何设计您的应用程序，使它支持一种以上语言或语系文字编辑的信息。

管理文件

当用户使用“存储”和“存储为”命令时，您通常将用户在当前活动窗口输入的数据写入一个文件。当用户使用打开命令选取一个文件时，您可以从文件中直接读取信息。您也可以使用文件管理器来读写文件。您还可以使用标准文件软件包提供的系统软件例程为用户提供存储和打开文件的标准的一致的界面。请参阅《Inside Macintosh 文件》(*Inside Macintosh: Text*) 一书中“文件管理器介绍”和“标准文件软件包”两章中有关这些方面的介绍。

分配存储器和启用进程

请参阅《Inside Macintosh 进程》(*Inside Macintosh: Process*) 一书的“进程管理器”一章中有关进程管理器如何执行您的应用程序的信息。请参阅《Inside Macintosh 内存》(*Inside Macintosh: Memorry*) 一书的“存储器管理器介绍”一章中关于系统软件如何管理存储器，您如何有效地管理您的应用程序存储器分区，以及您的应用程序如何分配、释放或操纵存储器的信息。

创建版本和引用程序

您的应用程序应该支持版本管理器功能，使用户能够在文稿间共享和自动地更新数据。请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书的“版本管理器”一章中有关支持出版和订阅功能的信息。

和其它应用程序通讯

系统软件提供了在应用程序间通讯的各种手段。您可以使用事件管理器的例程以高级事件的形式和其他应用程序通讯。高级事件不要求依附于特定的协议，所以他们的解释是由发送或接收他们的每个应用程序来定义的。苹果事件是高级事件，它遵循一个标准定义协议（苹果事件进程间通讯协议）。在大多数情况下，您应该使用苹果事件来实现应用程序间的通讯。因为苹果公司已经定义了一组标准的苹果事件，所有应用程序都可以用同样的方式解释特殊的苹果事件，并以可预期的方式做出响应。

事件管理器和苹果事件管理器都必须依靠程序间通讯（PPC）工具箱在应用程序间实际发送及接收事件。如果您需要得到事件管理器或苹果事件管理器不提供的附加控制部件或服务，您的应用程序也可以直接存取 PPC 工具箱。

如果您的应用程序支持出版和订阅功能，当有了引用程序可用的新数据或要求它创建一个新版本时，编辑管理器将向您的应用程序发送苹果事件通知。

请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中有关苹果事件、出版和订阅功能和对 PPC 工具箱进行直接存取的信息。

设计您的应用程序

如前所述，在您开始设计您的应用程序时，将需要广泛使用本书和《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书。当您了解了 Macintosh 应用程序的基本组成元素后，您就可以开始为您的应用程序添加独特的功能。我们相信，您必将发现《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 和《Inside Macintosh 系列丛书》中的其它图书对您创建应用程序是非常有价值的。

第二章　事件管理器

事件介绍 2-4
　　低级事件 2-8
　　操作系统事件 2-10
　　高级事件 2-13
　　事件优先级 2-15
　　转换上下文 2-15
关于事件管理器 2-16
事件管理器的使用 2-17
　　获取关于事件的信息 2-18
　　处理事件 2-21
　　　　调用 WaitNextEvent 函数 2-22
　　　　事件循环的编写 2-24
　　　　设定事件类型屏蔽参数 2-26
　　　　处理对话框中的事件 2-29
　　创建 'SIZE' 资源 2-30
　　处理低级事件 2-32
　　　　响应鼠标事件 2-33
　　　　响应键盘事件 2-38
　　　　扫描取消事件 2-46
　　　　响应更新事件 2-47
　　　　响应激活事件 2-50
　　　　响应磁盘插入事件 2-55
　　　　响应空事件 2-57
　　处理操作系统事件 2-58
　　　　响应暂停和恢复执行事件 2-60
　　　　响应鼠标移动事件 2-62
　　处理高级事件 2-67
　　　　响应来自其它应用程序的事件 2-69
　　　　搜索指定的高级事件 2-71
　　　　确定高级事件的发送程序 2-72

发送高级事件 2-73
请求回执 2-77
处理苹果事件 2-78
事件管理器参考资料 2-78
数据结构 2-79
事件记录 2-79
目标标识符记录 2-81
高级事件消息记录 2-82
事件队列 2-83
事件管理器例程 2-84
接收事件 2-84
发送事件 2-100
转换进程序号和端口名称 2-105
读鼠标 2-108
读键盘 2-110
获取计时信息 2-112
应用程序定义的例程 2-114
用于搜索高级事件队列的过滤函数 2-114
资源 2-115
'SIZE" 资源 2-115
事件管理器总结 2-120
Pascal 小结 2-120
常数 2-120
数据类型 2-122
事件管理器例程 2-123
应用程序定义的例程 2-124
C 小结 2-125
常数 2-125
数据类型 2-127
事件管理器例程 2-128
应用程序定义的例程 2-129
汇编语言小结 2-130
数据结构 2-130
陷阱宏 2-130
全局变量 2-131
结果代码 2-132

第二章

事件管理器

本章讲述了您的应用程序应如何使用工具箱的事件管理器来接收用户执行的动作的有关信息，如何接收应用程序在处理状态中的更改信息，以及如何与其它应用程序通讯。

例如，您可以从工具箱事件管理器中检索信息，该信息可为您的应用程序提供下列事件的详细情况：用户是否敲入一键或者按下鼠标按钮，您的应用程序的某个窗口是否需要更新，或一些涉及硬件相关或软件的动作是否要求您的应用程序响应。

同时，您的应用程序也可以使用事件管理器来支持Macintosh电脑中可用的协同多任务环境。该环境允许用户在多个已打开的应用程序之间进行交换并允许其它应用程序接受后台处理时间。通过使用事件管理器例程，可令系统软件在您的应用程序和其它应用程序之间协调安排处理时间。

事件管理器和进程管理器维持着协同多任务环境。进程管理器协调各应用程序的处理时间安排，而事件管理器则把应用程序处理状态的更改信息告诉您的应用程序。

请参阅《Inside Macintosh进程》(*Inside Macintosh : Processes*) 一书“进程管理器”一章关于进程管理器怎样安排应用程序执行时间的全面介绍。

您可以使用事件管理器与其它应用程序通讯。您的应用程序也可以使用苹果事件管理器的服务与其它应用程序通讯。

事件管理器和苹果事件管理器例程依靠程序间通讯 PPC）工具箱提供的服务，可以使您的应用程序与其它应用程序实现通讯。由事件管理器和苹果事件管理器所执行的服务满足了大多数应用程序的程序间通讯的需要。但是，如果希望获得事件管理器或苹果事件管理器没有提供的额外的控制部件或能力，您可以通过选择直接存取PPC工具箱。《Inside Macintosh应用程序间通信》(*Inside Macintosh : Interapplication Communication*) 一书“程序间通讯工具箱”一章介绍了您的应用程序可以使用的 PPC 工具箱例程。

若想了解事件管理器、苹果事件管理器和 PPC 工具箱提供的服务的信息，请参阅《Inside Macintosh : Interapplication Communication》一书。若想了解关于苹果事件的补充信息，其中包括关于怎样处理所需苹果事件的说明，请参阅《Inside Macintosh : Interapplication Communication》一书。

本章不仅介绍了工具箱事件管理器，而且还介绍了操作系统事件管理器。操作系统事件管理器保留着一个事件队列，该队列中储存着一些与硬件相关的事件，您可能希望您的应用程序响应这些事件。工具箱事件管理器会将操作系统事件管理器保留的信息告知您的应用程序。多数情况下，您的应用程序仅需与工具箱事件管理器对话即可。在本章中，“事件管理器”特指工具箱事件管理器。

本章是对事件的一个总体介绍，然后解释您应如何使用事件管理器来实现下列操作：

- 接收通过敲击键盘按键和鼠标按钮输入的应用程序信息
- 接收应用程序窗口需要被激活或更新的指示信息
- 当您的应用程序停止使用可用的系统资源时，允许其它应用程序使用这些资源
- 与其它应用程序通信

事件介绍

大多数Macintosh应用程序是通过事件在系统软件与应用程序，以及应用程序之间传送消息的，这些事件要求应用程序给予响应。事件就是一种手段，事件管理器通过它传递有关用户动作的信息、应用程序处理状态下的更改信息以及其它要求应用程序响应的事件信息。

事件管理器通过 EventRecord 类型的数据传送事件信息，其中包括事件类型的信息（例如，点按鼠标或击键），以及与事件相关的附加信息（例如，对于击键事件来说，还有按下哪一个键的信息）。

多数Macintosh应用程序是事件驱动应用程序。这就是说，它们会对接收到的事件作出响应，并根据事件的性质而采取不同的动作。特别是，Macintosh应用程序通过事件循环核查是否有事件发生，如果有，则对事件作出响应。如果没有事件需要处理，应用程序可以选择放弃占用CPU时间或在执行其它任务之前再次搜索事件队列。

您的应用程序一般从事件管理器中检索事件并通过使用WaitNextEvent函数放弃处理器时间。如果某个事件正在等待您的应用程序处理，WaitNextEvent函数将返回该事件。如果没有什么事件需要您的应用程序处理，那么WaitNextEvent函数就会把CPU时间分配给其他应用程序。

当有多个应用程序被打开时，用户可以随意指定当前活动的应用程序。当前活动的应用程序（或前台进程）是当前正与用户发生交互作用的应用程序。前台进程显示其菜单栏，并且它的窗口处在所有其它应用程序的前面。（进程这个术语是指一个已打开应用程序，比如一个已启用的桌面附件。）

在同一时刻只能有一个进程处于前台；但在后台可以存在多个进程。后台进程指的是当前不与用户对话的进程。前台进程具有存取CPU的第一优先权。其它进程只有在前台进程将CPU时间让与它们时它们才能存取CPU。

通过调用 WaitNextEvent 来检索事件，若无事件发生，您可以让其他后台应用程序在您的应用程序不占用CPU时间的时候获得CPU时间。当您的应用程序跑在后台时，它也可以在其它应用程序放弃CPU时获得处理时间。使用 WaitNextEvent还可以让用户在多个打开的应用程序之间切换。

处在后台的应用程序可以获得 CPU 时间，但不能与用户对话。(然而，用户可以将该应用程序置于前台－－例如点按该应用程序窗口。)当一个应用程序处于后台而又需要引起用户注意时，该应用程序也可以使用通知管理器发出一个通知请求。任何一个在'SIZE' 资源中设定了 canBackground 标志的应用程序，当它跑在后台时，都可以存取CPU。

一个进程（无论处于前台或后台）可按明确定义的次数在这两种状态间切换。

当您的应用程序将要暂停时，事件管理器将向它发送一个挂起事件(suspend)，当它又有处理时间并能恢复执行时，事件管理器又向它发送一个恢复事件（resume)，这样事件管理器就能保证在应用程序之间的切换以一种平滑的方式进行。事件管理器和进程管理器可在多个应用程序之间协调这种切换和 CPU 时间安排。

事件通常分为三类：低级事件、操作系统事件和高级事件。

当有下列一些情况发生时，事件管理器会将低级事件返回您的应用程序：例如用户按下鼠标按钮，释放鼠标按钮，敲击键盘上的一个键，或插入一张磁盘。您的应用程序需要激活或更新应用程序的某个窗口（刷新屏幕窗口内容)。当您的应用程序请求事件而没有任何其它事件可报告时，事件管理器将返回一个空事件。低级事件反映的是底层的硬件或软件现象。

操作系统事件反映应用程序进程状态的改变。例如，如果一用户将您的应用程序置于前台，进程管理器将通过事件管理器向您的应用程序发送一个事件。进程管理器和窗口管理器可以自动地完成重新激活应用程序的一些工作，您的应用程序被激活后，如果需要进行进一步处理，则必须小心慎谨。

高级事件用于不同的应用程序之间的通讯。

低级事件（除更新事件和空事件以外）和操作系统事件总是被引向前台进程。高级事件、更新事件和空事件则可以发送给前台进程或后台进程。

您可以通过设定事件类型屏蔽参数来指定您的应用程序接收的事件类型。事件类型屏蔽参数允许屏蔽不想接收的事件。例如，您可以接受除高级事件以外的所有事件。

事件的来源有很多：操作系统事件管理器，窗口事件管理器，进程管理器，以及PPC 工具箱。图 2 － 1 揭示的是工具箱事件管理器、系统软件的其它部分以及您的应用程序之间的相互关系。

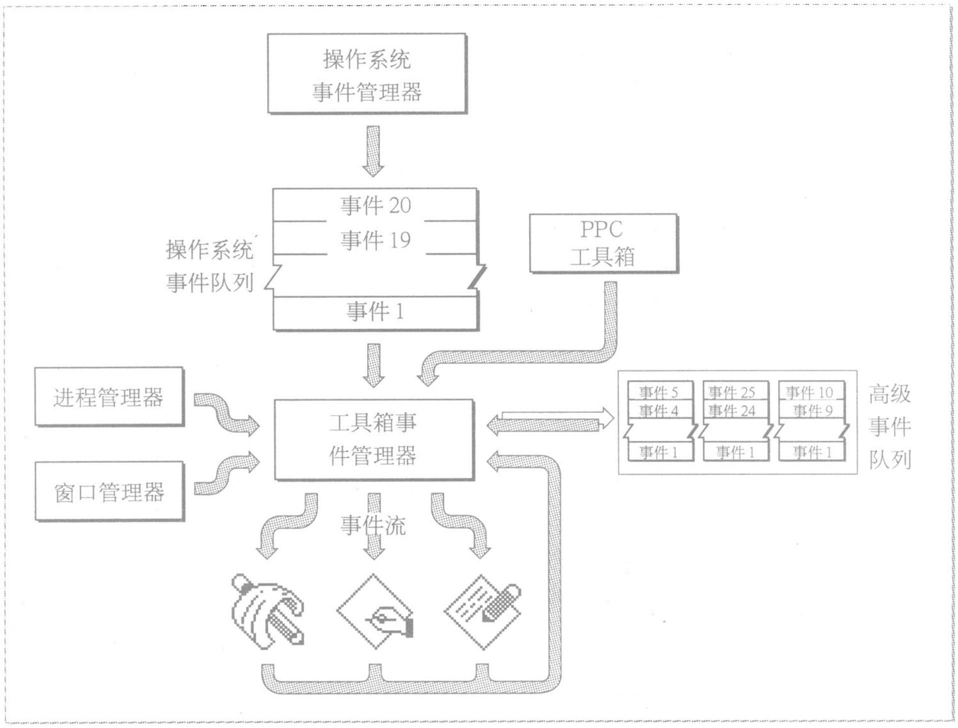


图 2 － 1　事件来源

操作系统事件管理器可以创建并保留一个操作系统事件队列。操作系统事件管理器检测并报告诸如鼠标敲点按、按键、磁盘插入等低级的硬件相关事件。并将这些事件排放在操作系统事件队列中。工具箱事件管理器在该事件队列中检索事件并把检索到的事件返回到您的应用程序，每次返回一个。

在操作系统事件队列中最多允许排列20个事件。当该事件队列排满时，操作系统事件管理器就会在发送事件的同时删除一些旧的事件以便为新事件留出空间。当必须删除某一事件时，操作系统事件管理器总是删除该队列中最陈旧的事件。但这种情况并不是经常发生的，您的应用程序处理事件的速度通常都能比用户生成事件的速度快得多。事件队列的实际容量由系统启动信息决定。请参阅《Inside Macintosh: File》中“文件管理器”一章：系统启动信息。

事件管理器也能报告来自窗口管理器和进程管理器的事件。如果一个窗口需要更新，激活，或失活，窗口管理器会向工具箱事件管理器发送一个事件。类似地，如果您的应用程序的处理状态发生了变化，进程管理器将向工具箱事件管理器发送一事件。工具箱事件管理器会向您的应用程序报告这些事件。

附　注:

System 6的 MultiFinder 具有 System 7 中进程管理器所提供的一些功能。如果运行未安装MultiFinder 的 System 6上，则仅支持一个单一应用程序环境。

您的应用程序可以使用事件管理器来发送和接收高级事件。如果希望在应用程序之间发送高级事件，事件管理器可以使用PPC工具箱代表您的应用程序。针对每个已打开的并能接收高级事件的应用程序，事件管理器可以单独保留一个队列，该队列被称作应用程序的高级事件队列，用以存放高级事件。应用程序高级事件队列的大小仅受可用存储器空间的限制。

应用程序的事件流仅由那些可供检索的事件组成。例如，对于后台应用程序来说，其事件流仅包含更新事件、空事件或高级事件。

当应用程序向事件管理器请求事件时，事件管理器根据事件的优先级返回下一个事件。总的说来，事件管理器是按下列优先级顺序返回事件的:

1. 低级事件
2. 操作系统事件
3. 高级事件

以下部分将更为详细地对低级事件，操作系统事件和高级事件加以说明。

低级事件

事件管理器利用低级事件来报告每个底层硬件或软件的事件发生情况。低级事件报告:

- 用户发出的动作（例如按下鼠标按钮，敲击键盘，或插入磁盘等）
- 屏幕上窗口的改变
- 空事件

报告用户动作的低级事件包括按下鼠标、释放鼠标、键按下、键释放、自动键和磁盘插入事件。当用户执行与每个事件相关的动作时，事件管理器将报告这些事件的一切内容。

按下鼠标和释放鼠标事件报告用户按下了或释放了鼠标按钮这个动作。对于这些事件，事件管理器将返回鼠标按钮被按下或释放时光标所处的位置信息。键按下和键释放事件报告用户按下了或释放了一个键。自动键事件报告用户已经按下一键并持续了一段时间。 对于键盘相关事件，事件管理器报告哪个键被按下。对于鼠标相关和键盘相关事件，事件管理器还报在事件发生时的同时告修正键（Option，Command，Caps Lock，Control，Shift 键）的状态。

当用户插入一张磁盘时，操作系统便会通过调用文件管理器函数PBMountVol在磁盘上安装卷。操作系统事件管理器随后生成一个磁盘插入事件。如果用户正在与一标准文件对话框进行人机对话，标准文件软件包将截获磁盘插入事件并进行处理。否则，事件将被搁置于事件队列中等待您的应用程序来检索。在大多数情况下，您的应用程序可以通过简单核查来显示是否成功地安装了卷，以便处理意外的磁盘插入事件。

低级事件中还包括更新事件和激活事件两种类型，事件管理器可以将其作为屏幕上窗口外观的更改结果报告给应用程序。例如，如果一个用户正在处理应用程序的几个打开文稿，用户可以通过点按适当的窗口，从一个文稿切换到另一个文稿窗口。您可以通过使用窗口管理器函数Findwindow 确定用户是否已点按了另一个窗口。如果用户点按了另一个窗口，您就可以使用窗口管理器过程SelectWindow生成相应的激活事件。在事件管理器向您的应用程序发送与该事件相关的任何激活事件之前，窗口管理器还将取消失活窗口的醒目显示，并醒目显示新激活的窗口。当您的应用程序发出下一个事件请求时，事件管理器将会返回一个激活事件。

激活事件告知该窗口是被激活或还是失活。您的应用程序应执行其它一切必要的动作以实现窗口从激活状态到失活状态的切换，反之亦然。例如，当一窗口处于激活状态时，您的应用程序应该显示该窗口滚动杆并还原一切选定项目。

激活事件实际上包含两个事件: 一个由失活窗口发出的事件,另一个由被激活的窗口发出的事件。

附　注:

如果用户在您的应用程序和另一个应用程序之间切换(例如通过点按另一个应用程序的窗口), 您的应用程序将负责激活或失活相应的窗口。如果您的应用程序在其 'SIZE' 资源中设定了 acceptSuspend ResumeEvents 和 doesActivateOnFGSwitch 标志, 您的应用程序便会通过操作系统事件得知该切换信息,并且当用户在应用程序之间切换时,您的应用程序不接受特殊的激活事件。

窗口管理器通过生成更新事件来控制屏幕上窗口的外观。窗口管理器始终监视着窗口放置的前后顺序并允许窗口重叠。当一个窗口覆盖另一个窗口,随后用户移动第一个窗口时,窗口管理器将生成一更新事件以使得新暴露出来的区域能被更新。事件管理器根据需要向那些窗口需要更新的应用程序报告更新事件。与其他低级事件不同的是,更新事件既可以发送给前台进程也可以发送给后台进程。

由窗口管理器生成的激活和更新事件不排放在操作系统事件队列而是被直接送到事件管理器。

当您的应用程序请求一事件时仅获取空事件, 您可以使用 WaitNextEvent 函数将时间让与其它进程。

前台应用程序接收到一个空事件时,您的应用程序如果在前台则可以做挂起处理(例如闪烁插入光标), 如果是后台程序则可执行其它任务。如果您希望您的应用程序在后台接收空事件, 那么您就必须在您的应用程序的 'SIZE' 资源中设定 canBackground 标志。如果您的应用程序在后台时不执行任何响应空事件处理, 则应设定cannotBackground标志。但是, 不管后台标志是如何设定的, 都可以接收更新事件 (如果在 'SIZE' 资源中设定了 isHighLevelEventAware 标志, 那么还可以接收高级事件)。

图 2 - 2 显示的是您的应用程序可接收的各种低级事件。请参阅从 2 - 32 页开始的 "处理低级事件" 部分内容, 以便获得您的应用程序应该如何响应低级事件的详细信息。

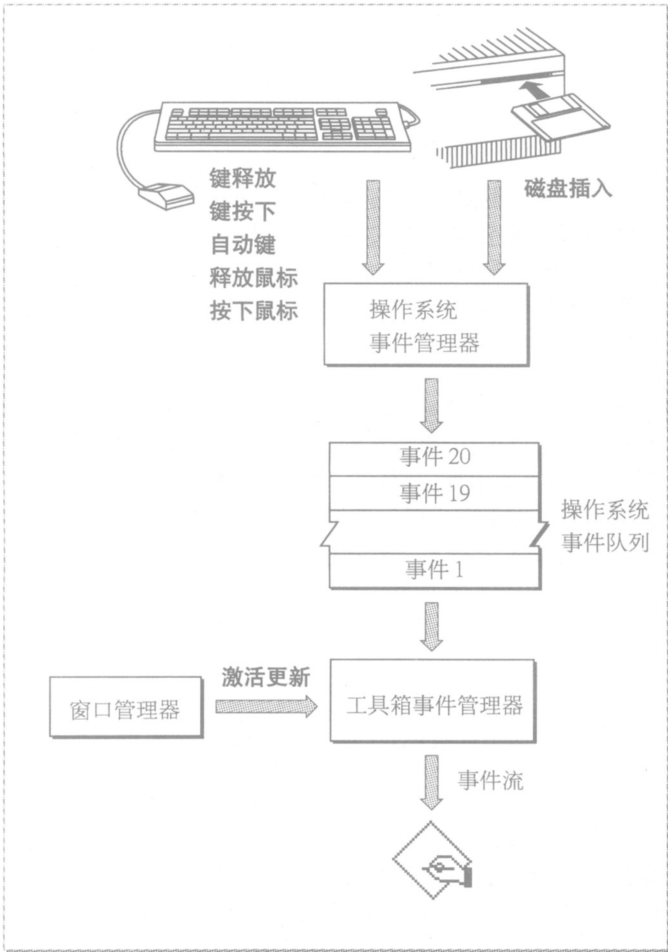


图 2-2　低级事件

操作系统事件

协同多任务环境允许用户与多个应用程序进行人机对话。进程管理器负责协调应用程序的处理时间安排，而事件管理器负责将应用程序操作状态信息通知其它应用程序。

例如，当您的应用程序即将被切换到后台时，事件管理器会向它发送一挂起事件。如果，您的应用程序又被切换回前台时，它将接收到一个恢复执行事件。挂起和恢复以及鼠标移动事件，都被称作操作系统事件。

图 2-3 解释了事件管理器是如何协助提供这种协同多任务环境的。进程管理器生成挂起，恢复和鼠标移动事件，由事件管理器报告给应用程序。

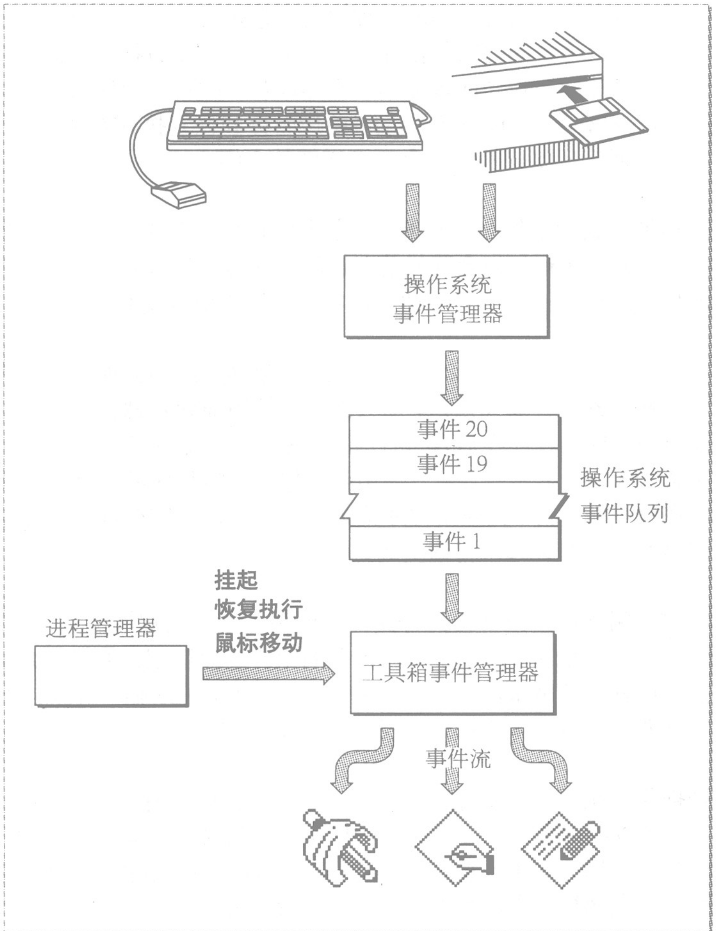


图 2–3　操作系统事件

附　注:

如果您的应用程序在其 'SIZE' 资源中设定了 acceptSuspendResumeEvents 和 doesActivateOnFGSwitch 标志，那么它在对操作系统事件作出响应时也会相应地激活或失活任务窗口。为了获得最大的兼容性，您的应用程序应该设定这些标志并处理挂起和恢复执行事件。请参阅始于 2 – 115 页的“大小资源”一节：关于 'SIZE' 资源中的这些和其它标志的详细信息。

当您的应用程序接收到一个挂起事件时，它只有在发出下一个请求以接收来自事件管理器的事件之后才能实实在在地交换到后台。在您的应用程序接收到挂起事件时，如果有必要，它应该将专用剪贴数据存储区转换为全局剪贴数据存储区。您的应用程序应该隐藏滚动杆，取消选定项的醒目显示，并且隐藏所有浮动窗口。如果您的应用程序显示剪贴板内容窗口，您也应该把这个窗口隐藏起来。然后，您应该调用 WaitNextEvent 来放弃占用 CPU 时间并允许操作系统调度其它进程。在对一挂起事件响应时，要尽量将处理时间降低到最小，这样用户会觉得电脑反应十分快捷。

当CPU时间控制权返回到您的应用程序时，它接收到的第一个事件是恢复执行事件。如果必要，应该将全局剪贴数据存储区转为专用剪贴数据存储区，并恢复其上下文信息。例如，您的应用程序应该显示所有滚动杆，醒目显示所有选定项，并显示所有浮动窗口。请参阅从2－60页开始的“响应暂停和恢复执行事件”一节：关于您的应用程序应该如何响应这些事件的全面而详细的介绍。

后台进程能接收的事件有更新事件、空事件和高级事件等。当您的应用程序在后台运行时，它不应该执行任何会降低前台进程响应速度的进程。当在后台接收到事件时，您的应用程序应该执行必要的动作以响应事件并快速返回。

您的应用程序千万不能在后台与用户进行人机对话。如果您的应用程序在后台执行时需要将一些特殊事件通知用户，那么您应该使用通知管理器来生成一个通知请求排队等候。当您的应用程序在后台进行时，您不要试图显示警告栏。但可以指定通知管理器发出语音警报、显示警告栏、使用代表您的应用程序的一个小图符和当前应用程序菜单的图符交替闪烁、在应用程序菜单中您的应用程序名称后面显示一个菱形图案等途径将这些动作组合起来付诸实施。

这些动作是用来向用户警告的方式，表明另一个应用程序需要引起用户的注意。使用通知管理器，您便遵守了赋予用户控制的用户界面原则，用户在方便的时候可以有选择地把应用程序的请求注意带到前台。请参阅《Inside Macintosh: Processes》一书“通知管理器”一章：关于如何发送通知请求的实例。

另一种操作系统事件是鼠标移动事件。每当光标移到当前WaitNextEvent函数的区域之外时，您可以请求事件管理器向您的应用程序发送一个鼠标移动事件。例如，当用户将光标从一文稿的文本区域移到滚动杆时，您可以将鼠标移动事件作为一种帮助您的应用程序改变光标外观的很方便的方法。请参阅从2－62页开始的“响应鼠标移动事件”一节：以了解更详细的实例。

高级事件

事件管理器可以提供一些例程,这些例程允许应用程序通过交换高级事件实现相互通讯。高级事件是指应用程序间通讯的信息，并从它那里接收某些信息。

例如,您的应用程序可以给另一个应用程序发送一个高级事件,命令它执行一个特定动作(诸如在电子表格中增加一行或更改段落的字体大小);还可以请求从那个应用程序获得信息－－例如,请求词典应用程序返回一个特殊单词的定义;您还可以将某些信息发送给那个应用程序,例如,您的应用程序可以使用一个高级事件向词典应用程序发送一个新词和定义列表。

附　注:
高级事件只能在7.0版或更高版本的系统软件中才能使用。

下一页中的图2－4显示了3个不同的应用程序通过发送和接收高级事件相互进行通讯的情形。事件管理器使用PPC工具箱发送高级事件。事件管理器为每个有资格接收高级事件的应用程序保留了一个高级事件队列。高级事件队列大小仅受可用存储器空间的限制。

若想在应用程序间进行有效的通讯，您的应用程序就必须定义它要响应的高级事件集,并让其它应用程序知道它能够接受的事件。如果具备了向其它应用程序发送或从其它应用程序接收事件的能力，您就可以通过程序间的通讯向您的用户提供更强大的功能。

总的来说,对于从一个应用程序向另一个应用程序发送一个高级事件请求的事件类型没有任何限定。但是如果希望一个应用程序发送的高级事件能够被另外一个应用程序理解，那么发送者和接收者必须达成一个协议，也就是解释事件的方法。苹果事件是高级事件，其结构和解释根据苹果事件进程间通讯协议（AEIMP）来确定。

您的应用程序应该支持所需苹果事件，在《Inside Macintosh 应用程序间通信》(*Inside Macintosh : Interapplication Communication*)一书中有相关的叙述。当您的应用程序被打开时，Finder使用所需苹果事件为它提供信息；当用户从Finder中打开或打印文件时，Finder还会告诉您的应用程序打开或打印的文稿名。

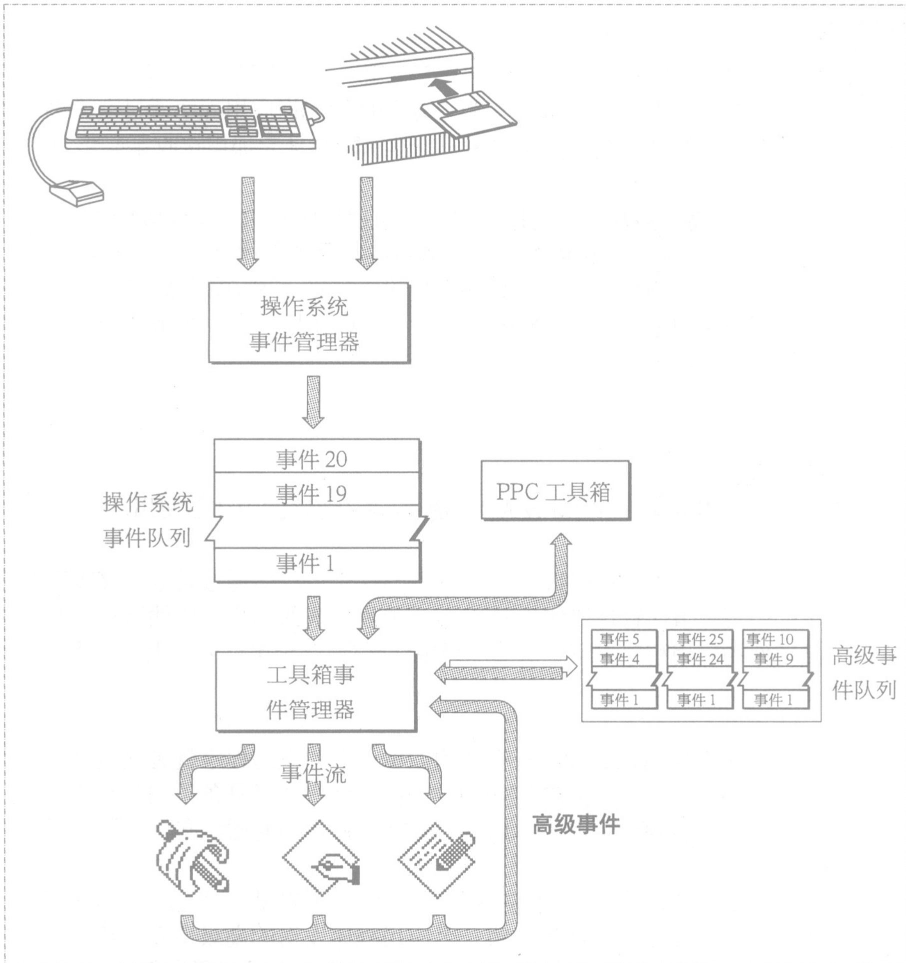


图 2-4　高级事件

另外，你也许会希望您的应用程序支持其它普通苹果事件。例如，编辑管理器使用苹果事件在各种应用程序之间传递关于文稿片段的信息，允许应用程序发布或引用文稿片段。编辑管理器向那些想要在它们的文稿内保留最新引用片段的应用程序发送适当的苹果事件。如果一个用户更换了一个已发布的文稿中的一个片段并更新编辑，编辑管理器会向应用程序发送一个苹果事件表示可以提供一个新的片段。应用程序在接收到苹果事件后可以根据用户的决定更新引用或对该事件不予理睬。若想全面了解响应编辑管理器发送事件的有关信息，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh : Interapplication Communication*) 一书“编辑管理器”一章。

如果要确保与其它Macintosh应用程序实现良好的兼容性和顺畅的对话，您应该尽可能地在高级事件中使用苹果事件协议。如果您的应用程序必须与使用不同协议的其它电脑上的应用程序通讯，或您的应用程序有其它特殊需要时，您应该规定新的协议。若想了解关于苹果事件和实现所需苹果事件集的详细情况，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh : Interapplication Communication*) 一书。

附　注:
所有发送和接收高级事件的 Macintosh 系统软件均使用苹果事件协议。

事件优先级

每类事件均有一特定的优先级。事件管理器根据下面的优先级顺序返回事件。

1. 激活事件
2. 按下鼠标，释放鼠标，键按下，键释放和按先进先出顺序（FIFO）的磁盘插入事件
3. 自动键事件
4. 更新事件（按窗口从前到后的顺序）
5. 操作系统事件（暂停、恢复执行、鼠标移动）
6. 高级事件
7. 空事件

有几个事件管理器例程可以限定在一种或多种特定类型的事件上运行。您可以把不希望接收的事件屏蔽掉。请参阅从 2 － 26 页的“设定事件类型屏蔽参数”部分以获得指定您想接收的事件类型的详细说明。

切换上下文

当前台进程调用 WaitNextEvent 或 EventAvail 请求事件，但未取到事件，进程管理器才有机会调度其它进程，后台进程方能获得 CPU 时间。后台进程应该定期让出 CPU 以保证必要时能及时返回到前台进程。

在 System 7（或早期版本的 MultiFinder）中，可用的 CPU 时间可以通过一个称作上下文切换（或仅仅是切换）的过程在多个进程间协调分配。所有切换均在明确规定的时间内即应用程序调用 WaitNextEvent 时发生。在上下文切换发生时，进程管理器便把 CPU 时间分配给一个没有接收过 CPU 时间的进程。可能发生的上下文切换有两种类型:主切换和次切换。

主切换是一个完全切换: 进程管理器把将被挂起的前台进程切换到后台，并向后台被调度的进程发送一个恢复执行事件（resume 事件），恢复其上下文信息，然后将其切换到前台，开始运行。

当进程管理器把CPU时间给予后台进程而又不把后台进程切换到前台时，这种切换则属于一个次切换。在一次切换中涉及的两个进程可以是两个后台进程或一个前台进程和一个后台进程。和在主切换中一样，进程管理器切换两个进程的A5区域和低速存储器环境。不管怎样，窗口的顺序并没有交换，两个进程都没有接收到挂起事件或恢复执行事件。

如果当前活动窗口是一个警告栏或一个定模式对话框时，只可以发生次切换。操作系统可以通过核查（在其他事情中）来了解当前活动窗口的窗口定义过程是不是dBoxProc，从而判断是否可以发生主切换，因为dBoxProc是特别为警告栏和定模式对话框保留的。（如果当前活动窗口是一个可移动定模式对话框，那么主切换仍然可以发生。）

附 注:
如果您的应用程序调用的是一个作出事件调用命令的系统软件例程，它还可以自行断开。例如，当您的应用程序调用 ModalDialog 时，就可能会发生一个次切换。

后台进程可以接收CPU时间并在后台执行任务，却不应与用户对话或执行那些会导致前台进程响应速度下降的任务。

您的应用程序可以通过设定'SIZE'资源中的标志来向操作系统指示其时间安排选项，诸如应用程序在后台时是否可以使用空事件处理时间，它是否可以接受挂起和恢复执行事件等等。在System 7中执行的每个应用程序，以及在带有 MultiFinder 的 System 6中执行的每个应用程序，都应包含一个 'SIZE' 资源。请参阅从2 － 30页开始的“创建 'SIZE' 资源”一节中关于怎样设定该信息的详细说明。

关于事件管理器

工具箱事件管理器提供一系列例程，可将关于用户操作的信息、进程状态信息等传送给应用程序，并使应用程序可以与其它应用程序进行通讯。

本章的余下部分阐述了下列问题:

- 如何构造主事件循环，以便接收和处理事件
- 如何创建 'SIZE' 资源以指定您的应用程序的存储器环境和 CPU 时间安排选项
- 如何响应多数类型的事件
- 如何接收和处理高级事件
- 如何向其它应用程序发送高级事件

事件管理器的使用

您可以使用事件管理器来接收硬件相关事件,或关于您的应用程序窗口的外观变化信息,或关于您的应用程序的工作状态改变的信息。您也可以使用事件管理器来直接与其它应用程序通讯。这种通讯包括向其它应用程序发送事件、从其它应用程序接收事件以及从其它应用程序搜索特定事件。

您的应用程序既可以发送又可以接收高级事件,但它通常只接收低级事件而不发送它们。您的应用程序用同样的方法接收低级事件、操作系统事件和高级事件,即通过向事件管理器请求下一个可用的事件来实现。如果您的应用程序接收的是一个高级事件,您的应用程序可能会需要使用另一个事件管理器或苹果事件管理器例程来检索事件和伴随该事件的附加信息。

在使用事件管理器之前,您可以使用Gestalt函数来判定事件管理器的某些功能是否可用。请参阅《Inside Macintosh操作系统实用程序》(*Inside Macintosh: Operating System Utilities*) 一书的 "Gestalt Manager" 一章关于Gestalt函数的介绍。

如果您的应用程序发送或接收高级事件,您应该使用Gestalt函数和gestalt-PPCToolboxAttr选择程序来确定PPC工具箱是否存在。使用Gestalt函数和gestaltOSAttr选择程序来确定进程管理器是否可用。如果PPC工具箱和进程管理器存在,那么系统软件将为高级事件提供支持。

如果您的应用程序发送或接收苹果事件,使用Gestalt函数和gestaltAppleEventsAttr选择程序来确定苹果事件管理器是否可用。

您的应用程序在使用事件管理器之前需要初始化QuickDraw,字体管理器和窗口管理器。您的应用程序可以使用InitGraf, InitFonts, 和InitWindows过程来完成这项初始化工作。

当您的应用程序启动时，您可以调用FlushEvents过程来清除操作系统事件队列中另一应用程序剩下的没被处理的低级事件。例如，您也许想删除当Finder启用您的应用程序时，用户输入的按下鼠标或敲击键盘事件。

这一部分介绍怎样从事件管理器中检索事件，怎样屏蔽不想要的事件，怎样规定您的应用程序的存储器和时间安排选项，怎样处理接收来自事件管理器的每类事件。

获取关于事件的信息

您可以通过事件记录获取关于事件的信息。EventRecord数据类型定义了事件记录，其结构如下结构:

```
TYPE      EventRecord =
          RECORD
               what:     Integer;        {event code}
               message:  LongInt;        {event message}
               when:     LongInt;        {ticks since startup}
               where:    Point;          {mouse location}
               modifiers:Integer;        {modifier flags}
          END;
```

域说明

what

what域指出了所接收事件的类型。事件类型可以根据以下常数来识别:

```
CONST
nullEvent      = 0; {no other pending events}
mouseDown      = 1; {mouse button pressed}
mouseUp        = 2; {mouse button released}
keyDown        = 3; {key pressed}
keyUp          = 4; {key released}
autoKey        = 5; {key repeatedly held down}
updateEvt      = 6; {window needs updating}
diskEvt        = 7; {disk inserted}
activateEvt    = 8; {activate/deactivate window}
osEvt          = 15;{operating-system event-- }
                    { resume, suspend, or }
                    { mouse-moved}
kHighLevelEvent= 23;{high-level event}
```

message

message 域包含与事件相关的附加信息。对于该信息的解释取决于事件类型。现将每个事件类型的 message 域的内容总结如下:

事件类型	事件消息
空、释放鼠标、按下鼠标	未定义
键释放 键按下	低位字中的字符代码和模拟键代码。对于苹果桌面总线(ADB)键盘，高位字的低位字节中含有键盘事件发生处的键盘地址。
自动键	高位字的高位字节保留。
更新、激活	指更新、激活或失活窗口的指针。
磁盘插入	低位字为驱动器号码，高位字为文件管理器结果代码。
恢复执行	suspendResumeMessage 常数在位 24-31之间，位0中的1表示事件是一个恢复执行的事件。而位1中的1或0表示剪贴板转换是否需要，位2-23保留。
挂起	suspendResumeMessage 常数在位24-31之间，位0中的0表示事件是一个挂起的事件。位1未定义，位2-23保留。
移动鼠标	mouseMoveMessage 常数在位 24-3 之间，位2-23保留，位0和位1未定义。
高级事件	高级事件所属的事件等级。高级事件的message和where域定义了被接收的高级事件的具体类型。

when

when 域表示事件发送的时间 从系统启动开始计时)。如果需要，您可以使用 when 域来比较两个相互衔接的鼠标事件间的时间差。

where

对于低级事件和操作系统事件，where域包含光标在事件投寄时的位置(以全局坐标表示)。对于高级事件，where域包含一个第二事件标识符。事件标识符在高级事件的message域定义的事件等级中定义了事件的特定类型。对于高级事件，您应该将where域解释为具有OSType数据类型，而不是 Point 数据类型。

`modifiers` modifiers 域包含修正键和鼠标按钮在事件投寄时的状态信息。对激活事件，该域还表示窗口是被激活还是失活的信息。在 System 7 中，它还表示一个点按鼠标事件是否将您的应用程序切换到了前台。

每个修正键在事件记录的 modifiers 域中都用一个特定位来代表。图 2-5 说明了如何解释modifiers 域。您可以检验事件记录的 modifiers 域以确定在事件执行时按了哪一个（如果有的话）修正键。修正键包括 Option 、Command、Caps Lock、Ctrl和 Shift 键。如果您的应用程序赋予了这些键中任一个键（它和其它键组合在一起）特殊的意义，或当鼠标按钮按下时，您可以测试 modifiers 域的状态以确定应用程序应该执行的动作。例如，您可以使用这个信息来确定用户是否同时按了Command键和另一个键来进行菜单命令项目选择的操作。

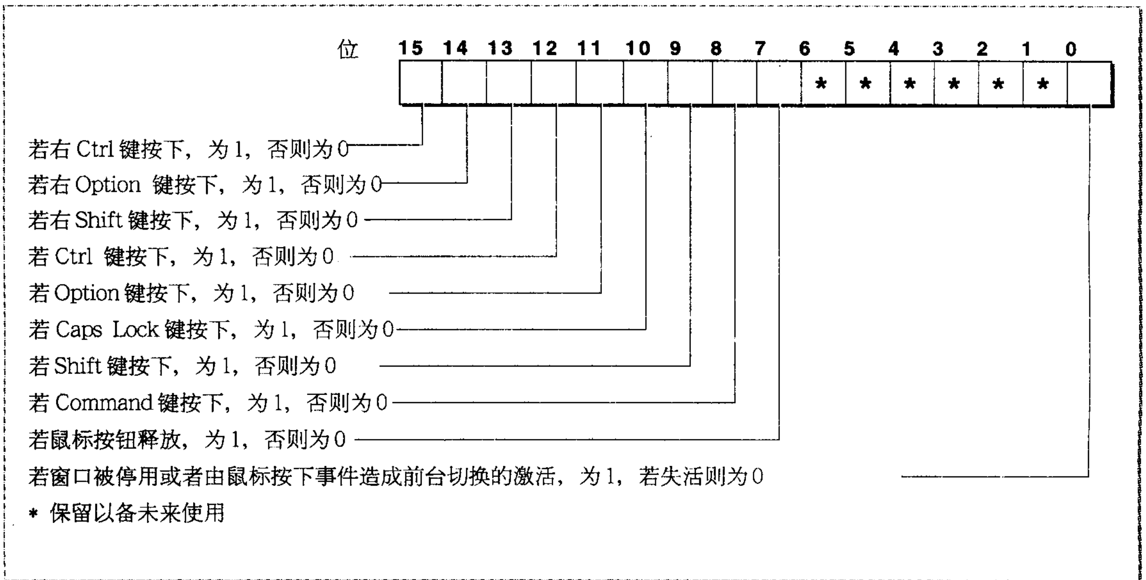


图 2-5 事件记录的 modifiers 域

modifiers域中的位0能够给出附加信息，但这只有当事件是激活事件或者是一个鼠标按下事件时才有效。

对激活事件而言，如果事件消息指向的窗口应该被激活，那么位0的值为1；如果窗口失活，其值为0。

对于System7的鼠标按下事件，位0表示一个鼠标按下事件是否已将您的应用程序切换到了前台。如果是，那么位0的值为1，否则为0。

您也可以用以下这些常数来测试modifiers域中各个不同位的设定状态:

```
CONST activeFlag  = 1;      {set if window being activated or if }
                            { mouse-down event caused fgnd switch}
      btnState    = 128;    {set if mouse button up}
      cmdKey      = 256;    {set if Command key down}
      shiftKey    = 512;    {set if Shift key down}
      alphaLock   = 1024;   {set if Caps Lock key down}
      optionKey   = 2048;   {set if Option key down}
      controlKey  = 4096;   {set if Control key down}
```

注意: 当鼠标按钮释放时Modifiers域中的对应位的值被设定。修正键的位在相应的键按下时也被设定。

一些键盘不区分Ctrl、Shift和Alt键的左右位置; 例如左右Shift键的模拟键代码可能是相同的。对于这些键盘，如果用户按下Ctrl、Shift或Alt键，事件管理器只设定和shiftKey、optionKey和controlKey常数相对应的位。对于那些区分左右键的键盘，事件管理器设定 modifiers 域中的位来表示按下的是左边的Ctrl、Shift或Alt键还是右边的Ctrl、Shift或Alt键。例如，当用户按下了右Shift键时，事件管理器设定modifiers 域中的位13的值为1，如果用户按的是左Shift键，设定的则是位9。在大多数情况下，您的应用程序不需要区分左右Ctrl、Shift和Option键。

处理事件

应用程序通过向事件管理器请求下一个事件,而且每次只接收一个事件。事件管理器通过EventRecord类型的数据来传送事件信息。您也可以把事件记录做为一个参数利用事件管理器的例程来接收（或在 EventAvail 的情况下，只是简单地查看）等待您的应用程序处理的下一个可用事件。

您的应用程序可以调用WaitNextEvent函数从事件管理器中检索事件。如果没有事件等待您的应用程序处理，WaitNextEvent函数便将CPU时间分配给其它应用程序。如果有一个事件正在等待您的应用程序处理，WaitNextEvent函数会将该事件返回并将返回的事件从您的应用程序的事件流中删除。

EventAvail函数获取下一个特殊类型的可用事件时，仅将它返回您的应用程序，却不将其从您的应用程序的事件流中删除。这就是为什么说在EventAvail的情况下，接收事件的过程只是简单地查看，而无须实际处理该事件。

附注 GetNextEvent 函数的功能同 WaitNextEvent 函数较类似，前者为 System 6.0 所用，后者支持 System 7.0 的多任务环境。

调用 WaitNextEvent 函数

这一节着重讨论 WaitNextEvent 函数本身您的应用程序通常重复调用 WaitNextEvent 函数。下一节“事件循环的编写”介绍如何与其它例程一起使用 WaitNextEvent 函数来处理事件。

WaitNextEvent 函数要求 4 个参数:

- 事件类型屏蔽参数（eventMask）
- 事件记录（theEvent）
- 休眠值（sleep）
- 鼠标区域（mouseRgn）

如果有事件发生 WaitNextEvent 函数将返回包含被检索事件信息事件记录类型数据。

eventMask 参数指定了您接收的事件的类型。WaitNextEvent 按事件优先级的次序，一次返回一个事件。如果您的应用程序屏蔽了它不想接收的事件类型，这类事件就不会在您的应用程序请求事件时返回给它。但是这类事件并不从事件流中删除（如要从操作系统事件队列中删除事件，您可以使用一个带屏蔽参数的 FlushEvent 过程，该屏蔽参数仅指定那些您要从队列中删除的事件）。请参阅 2-26 页的“设定事件类型屏蔽参数”中有关如何使用常数来设定 eventMask 参数值的实例。

sleep 参数指定了在没有事件等待您的应用程序处理的情况下，该程序同意释放 CPU 的时间（以 Tick 为单位）。当时间到了或有了等待您的应用程序处理的事件，进程管理器会安排您的应用程序来处理。一般来说，sleep 参数应该指定为一个大于 0 的值，使其它应用程序可以在需要时占用 CPU 时间。例如，当用户在编辑文字，而您的应用程序需要周期性地闪烁插入光标，或使用 TextEdit 来闪烁插入光标时，您的应用程序指定的值不应该大于 GetCaretTime 函数返回的值。

用 mouseRgn 参数指定一个屏幕区域，该区域采用全局坐标系，一旦鼠标移出该区域，事件管理器生成一个鼠标移动事件。当您的应用程序接收到一个鼠标移动事件时，它应重新计算 mouseRgn 参数；若您传送的是一个空区域或 NIL 区域间接指针，事件管理器则不返回鼠标移动事件。mouseRgn 参数可作为更改光标形状的简单方法，例如，当用户将光标从窗口的内容区移到滚动杆时光标形状即会发生变化。请参阅 2-62 页的“响应鼠标移动事件”中有关如何设定和更改 mouseRgn 参数的内容。

程序清单 2-1 给出了使用 WaitNextEvent 函数的一个实例。

程序清单 2-1 使用 WaitNextEvent 函数

```
VAR
    eventMask:          Integer;
    event:              EventRecord;
    cursorRgn:          RgnHandle;
    mySleep:            LongInt;
    gotEvent:           Boolean;

    eventMask := everyEvent;    {accept all events}
    mySleep := MyGetSleep;      {set an appropriate sleep value}
    cursorRgn := MyGetRgn;      {set the region as appropriate}
    gotEvent :=
WaitNextEvent(eventMask,event,mySleep,cursorRgn);
```

程序清单 2-1 中的代码段指定 WaitNextEvent 应该返回等待处理的下一个事件，事件类型没有限制，若没有这样的事件就放弃 CPU 时间，若用户将光标移出指定区域 WaitNextEvent 就返回一个鼠标移动事件。

WaitNextEvent 函数只有在检索到一个事件后，或者 sleep 时间到期后才开始返回。如果没有 eventMask 参数指定类型的事件（除了空事件外）等待您的应用程序的处理，或者 sleep 参数指定的时间未到期，WaitNextEvent 可以将 CPU 时间分配给后台进程。

如果 WaitNextEvent 检索到的不是一个空事件，它将返回一个 TRUE 值。如果没有 eventMask 参数指定类型的事件等待应用程序处理（除了空事件外），WaitNextEvent 将返回 FALSE。

在向您的应用程序返回一个事件之前，WaitNextEvent 可执行其它任务甚至截取该事件。WaitNextEvent 函数可以执行下列任务:

- 调用操作系统事件管理器函数 SystemEvent 来确定检索到的事件应该由您的应用程序处理还是应该由操作系统来处理。例如，若事件是一个 Command—Shift—数字键序列，就由事件管理器截取该事件并调用相应的‘FKEY’资源来执行有关的动作
- 如果设定了闹钟报警且已到是闹钟报警时间，可以结束闹钟报警。用户可以使用闹钟桌面附件设定报警功能

- 调用 SystemTask 过程，该过程可以给每个已打开的桌面附件或设备驱动程序提供执行任何周期动作的时间。桌面附件或设备驱动程序指定了周期动作发生的频率，SystemTask 会在适当的时间间隔给桌面附件或设备驱动程序提供执行时间

在 System 7 中，WaitNextEvent 函数在下列情况下，将向您的应用程序报告一个挂起事件:

- 您的应用程序跑在前台，而且用户从苹果菜单中打开了一个桌面附件或其他项目
- 用户点按了属于桌面附件或另一个应用程序的窗口，或者
- 用户从应用程序菜单中选择了另一个进程

当您的应用程序被切换到后台，事件管理器将事件（除了您的应用程序能够在后台接收的事件外）引向新激活的进程，直至用户将您的应用程序或其它应用程序切换回前台。

事件循环的编写

在一个事件驱动应用程序（即通过接收和响应事件决定在什么时候做什么事情的应用程序）中，您可以通过调用事件管理器的例程得到关于等待处理事件的信息。您可以采用一种循环方式向事件管理器请求重复调用这些例程，我们称之为“事件循环”。

程序清单 2-2 给出了一个运行在 System 7 上的简单的应用程序事件循环（MyEventLoop 的过程）。

程序清单 2-2 事件循环

```
PROCEDURE MyEventLoop;
VAR
    cursorRgn:          RgnHandle;
    gotEvent:           Boolean;
    event:              EventRecord;
BEGIN
    cursorRgn := NewRgn;{pass an empty region the first time thru}
    REPEAT
        gotEvent := WaitNextEvent(everyEvent, event, MyGetSleep,
                                  cursorRgn);
        IF (event.what <> kHighLevelEvent) AND (NOT gInBackground)
           THEN MyAdjustCursor(event.where, cursorRgn);
        IF gotEvent THEN  {the event isn't a null event, }
           DoEvent(event) { so handle it}
        ELSE              {no event (other than null) to handle }
           DoIdle(event); { right now, so do idle processing}
    UNTIL gDone;          {loop until user quits}
END;
```

MyEventLoop 过程重复调用 WaitNextEvent 来检索事件。若没有（除了空事件之外）等待应用程序处理的事件，WaitNextEvent 函数返回一个 FALSE 布尔值。若检索到的是一个非空事件，就返回 TRUE。

在 WaitNextEvent 返回后，MyEventLoop 过程首先判断该事件是否属于非高级事件以及程序是否跑在前台，若满足判断条件则调用 MyAdjustCursor 过程在必要时调整光标。您调整光标通常是为了响应一个鼠标移动事件以及其它事件。如果应用程序接收到的是一个非高级事件，这个代码段每次通过事件循环对光标调整一次。若接收的是一个高级事件或该程序跑在后台，该代码段不对光标进行调整，因为高级事件的 where 域包含事件标识符，而不是光标的位置信息。而且光标的外观仅由前台进程负责设置。

如果 WaitNextEvent 检索到的是非空事件，事件循环调用 DoEvent 来处理事件。否则，该过程就调用应用程序定义的空闲过程 DoIdle（DoEvent 过程和 DoIdle 均为应用程序自定义的过程）。

附 注

如果您的应用程序使用不定模式对话框，您就需要适当地处理发生在其中的事件。您可以选择调用和在其它窗口中处理事件的例程一样的例程来处理不定模式对话框中的事件；这正是我们在本章中所使用的方法。您也可以选择调用事件循环中的 IsDialogEvent 函数。请参阅 2-29 页“在对话框中处理事件”部分，以便获得有关在警告栏、定模式对话框、可移动定模式对话框和不定模式对话框中处理事件的信息。请参阅本书“对话管理器”一章中有关方面的更详细信息。

如果希望您的应用程序不仅可以在单一任务环境下（例如没有 MultiFinder 的 System 6）运行，而且可以在多任务环境下运行，那么在您的事件循环一开始就应该对运行环境进行测试，从而决定调用什么函数来检索事件。如果在单任务环境下，应调用 GetNextEvent 函数来检索事件。同时调用 SystemTask 以使得桌面附件能够执行周期动作。反之则调用 WaitNextEvent 函数，这时就不应该调用 SystemTask 过程。

程序清单 2-2 所示的事件循环调用的是一个应用程序定义的过程 DoEvent，通过它来确定调用 WaitNextEvent 检索到的是哪一类事件。程序清单 2-3 定义了一个简单的 DoEvent 过程。该 DoEvent 过程是通过检验事件记录中 what 域的值来确定接收的事件类型的，然后调用一个适当的应用程序定义的例程来进一步处理该事件。

程序清单 2-3 处理事件

```
PROCEDURE DoEvent (event: EventRecord);
VAR
    window:       WindowPtr;
    activate:     Boolean;
BEGIN
    CASE event.what OF
        mouseDown:
            DoMouseDown(event);
        mouseUp:
            DoMouseUp(event);
        keyDown, autoKey:
            DoKeyDown(event);
        activateEvt:
            BEGIN
                window := WindowPtr(event.message);
                activate := BAnd(event.modifiers, activeFlag) <> 0;
                DoActivate(window, activate, event);
            END;
        updateEvt:
            DoUpdate(WindowPtr(event.message));
        diskEvt:
            DoDiskEvent(event);
        osEvt:
            DoOSEvent(event);
        kHighLevelEvent:
            DoHighLevelEvent(event);
    END; {of case}
END;
```

下一节介绍如何设定事件类型屏蔽参数，
程序的 'SIZE' 资源。以后各节将给出处理每一类事件的代码段。

设定事件类型屏蔽参数

通过设定事件类型屏蔽参数您可以限定几个事件管理器的例程只操作一个特定的事件类型或一组事件类型。您也可以指定应用程序可接收的事件类型，从而使您不想接收的事件类型失效（或"屏蔽掉"）。

事件类型屏蔽参数是一个整数，对于每种类型的事件对应其中的一位。如果设定某一位为1，那么，事件管理器将返回该类型事件。如果将此位设定为 0，事件管理器将不返回该类型事件。如果需要接收所有类型的事件，只需将事件类型屏蔽参数的每一位均设定为 1 即可，即常数 everyEvent 。

```
CONST  everyEvent      =   -1;       {every event}
```

图 2-6 显示了事件类型屏蔽参数中与每种事件类型相对应的位。

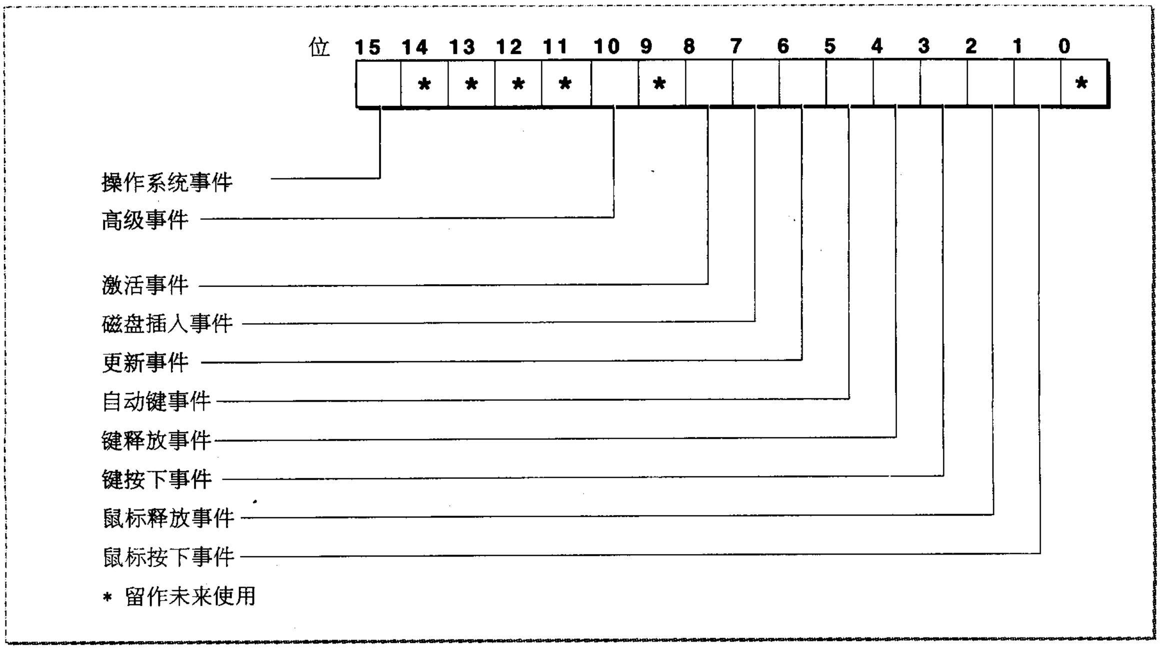


图 2-6　事件类型屏蔽参数

以下为定义的每一类事件类型的常数:

```
CONST  mDownMask            =  2;     {mouse-down event          (bit 1)}
       mUpMask              =  4;     {mouse-up event            (bit 2)}
       keyDownMask          =  8;     {key-down event            (bit 3)}
       keyUpMask            =  16;    {key-up event              (bit 4)}
       autoKeyMask          =  32;    {auto-key event            (bit 5)}
       updateMask           =  64;    {update event              (bit 6)}
       diskMask             =  128;   {disk-inserted event       (bit 7)}
       activMask            =  256;   {activate event            (bit 8)}
       highLevelEventMask   =  1024;  {high-level event          (bit 10)}
       osMask               =  -32768;{operating-system event    (bit 15)}
```

通过加或减这些常数，您可以选定任意事件子集。例如，您可以使用以下代码段实现只接收高级事件和鼠标按下事件:

```
myErr := WaitNextEvent(highLevelEventMask + mDownMask, myEvent,
                       mySleep, myMRgnHnd);
```

everyEvent 常数表示您希望接收每一类事件。您可以使用下列代码段来接收除了释放鼠标以外的所有事件:

```
myErr := WaitNextEvent(everyEvent - mUpMask, myEvent, mySleep,
                       myMRgnHnd);
```

某一类型的事件被屏蔽，并不表示这一类事件从事件流中删除掉。如果一种类型的事件被屏蔽了，只表明事件管理器在报告事件流中的事件时忽略它们。注意，事件类型屏蔽参数不能屏蔽空事件。若没有等待处理的事件，事件管理器总是返回一个空事件。但是，如果您不希望事件管理器在您的应用程序跑在后台时向它报告空事件，您可以在应用程序的‘SIZE’资源中设定 cannotBackground 标志。

在大多数情况下，您应该总是使用 everyEvent 作为您的事件类型屏蔽参数。用户期望大多数应用程序能够对键盘、鼠标、更新及其它事件响应的。

返回到您的应用程序的事件类型也受到系统事件类型屏蔽参数的影响。事件管理器会为每个应用程序保留一个系统事件类型屏蔽参数。低级事件类型获得的系统事件类型屏蔽参数控制函数是在操作系统事件队列中传递的。事件管理器在决定何种低级事件发送到操作系统队列时，它使用当前执行的进程（当前在执行的进程即和 CurrentA5 全局变量相关的进程）的系统事件类型屏蔽参数。系统事件类型屏蔽参数是一个整数，每一位都对应着相应的低级事件类型。以下为定义好的每一个低级事件的常数:

```
CONST mDownMask      = 2;    {mouse-down      (bit 1)}
      mUpMask        = 4;    {mouse-up        (bit 2)}
      keyDownMask    = 8;    {key-down        (bit 3)}
      keyUpMask      = 16;   {key-up          (bit 4)}
      autoKeyMask    = 32;   {auto-key        (bit 5)}
      diskMask       = 128;  {disk-inserted   (bit 7)}
```

当低级事件（除了更新或激活事件外）发生时，如果该事件未被当前执行进程的系统事件类型屏蔽参数屏蔽掉，操作系统事件管理器才将它发送到操作系统事件队列中。在应用程序启动后，操作系统会将导程序的系统事件类型屏蔽参数初始化，其初值设定操作系统事件队列可发送释放鼠标、按下鼠标、键按下、自动键和磁盘插入等事件。这样，系统事件类型屏蔽参数的初始设定状态是:

```
systemEventMask := everyEvent - keyUpMask;
```

应用程序一般不应该更改系统事件类型屏蔽参数，除非它需要对键释放事件响应 大多数应用程序忽略键释放事件）。若您的应用程序需要接收键释放事件，您可以调用操作系统事件管理器过程 SetEventMask 来更改系统事件类型屏蔽参数。注意，若您的应用程序不是当前进程，它就不能指望接收键释放事件。例如，虽然您的应用程序是前台 和当前）进程，进行了一个次切换，事件管理器在传递低级事件类型时便会使用后台进程 现在成为当前处理进程）系统事件类型屏蔽参数。当您的应用程序再次变成当前进程时，事件管理器在传递低级事件时才会使用您的应用程序的系统事件类型屏蔽参数。

处理对话框中的事件

若您的应用程序使用警告栏、定模式对话框、可移动定模式对话框或不定模式对话框，您需要确保您的应用程序能够适当地处理这些对话框中的事件。

为了在警告栏中显示和处理事件，您可以调用 Alert、NoteAlert、CautionAlert 和 StopAlert 等对话管理器函数。对话管理器要在用户点按按钮 通常是“确定” 或 “ 取消”按钮）前处理用户生成的所有事件的。当用户点按了确定或取消按钮后，警告栏函数醒目显示被点按的按钮，关闭警告栏，并向您的应用程序报告用户的选择。您的应用程序执行和该按钮有关的适当动作来对此进行响应。

对于定模式对话框，您可以调用对话管理器过程 ModalDialog。对话管理器在用户选定一个控制部件前，需要处理大多数交互动作。一旦用户选择了一个有效部件，ModalDialog 过程就向应用程序报告，应用程序就执行和该部件有关的动作进行响应。您的应用程序通常重复调用 ModalDialog，对 ModalDialog 报告的被点按有效部件做出响应，直至用户选择了确定或取消按钮为止。对于警告栏和定模式对话框，您还应该提供一个事件过滤函数，作为给警告栏函数或ModalDialog 过程的一个参数。当用户和警告栏或定模式对话框对话时，首先将事件传送给事件过滤函数。您的事件过滤函数可以处理那些未被对话管理器处理的事件，或者，必要时可以有选择地处理通常是由对话管理器处理的事件。请参阅本书中“对话管理器”一章中有关警告栏和定模式对话框的过滤函数的信息。

为了处理可移动定模式对话框中的事件，您可以使用对话管理器函数 IsDialogEvent 和 DialogSelect，或者您可以不用对话管理器，而用其它工具箱例程来处理这些事件。

对于不定模式对话框，您可以选择处理其中的事件，而方法可以是和处理其它窗口中的事件相类似的方法；也就是说，当您接收到一个事件后，应首先确定所发生事件的类型，然后根据前台窗口的类型采取适当的动作。如果一个不定模式对话框处在前台，您就可以提供对不定模式对话框采取特殊动作的代码段，并调用DialogSelect函数来处理您的代码段没有处理的事件。这正是在本章中所使用的方法。另外，您也可以选择调用事件循

环中的IsDialogEvent函数。如果这样做，您就可以调用IsDialogEvent函数来确定涉及一个不定模式对话框的事件是否属于您的应用程序。如果该事件涉及一个不定模式对话框（包括空事件）并且不定模式对话框处于激活状态，那么IsDialogEvent就会回送TRUE。否则IsDialogEvent回送 FALSE。

如果IsDialogEvent返回的是TRUE，您的应用程序就可以核查并显示发生了哪一类事件。根据事件的类型，它可以选择处理该事件。

无论您使用哪种方法，只要您的应用程序选择后决定不处理该事件，那么它应该调用DialogSelect。DialogSelect 函数能够处理不定模式对话框事件（包括空事件）。而且必要时，它还会在可编辑文字项目中闪烁插入光标。

有关 DialogSelect 函数和对话框事件的更详细信息，请参阅本书的“对话管理器”一章。

创建 'SIZE' 资源

您的应用程序应该包括一个 'SIZE' 资源。您需要使用‘SIZE’资源告知操作系统您的应用程序对存储器空间的需求。这样，操作系统就可以为您的应用程序设置一个大小适当的（存储器）分区。另外，您还可以使用‘SIZE’资源为操作系统指示某些安排选项，例如您的应用程序是否能够接受挂起和恢复执行的事件。

您也可以在 System 7 的‘SIZE’资源中指定附加信息，指示您的应用程序是否是真正的的 32 位应用程序，是否支持出版文稿，是否使用 TextEdit 在线输入服务，是否希望接到它启动的应用程序运行结束的通知，以及是否希望接收高级事件等。

'SIZE'资源由 16 位的标志域和后面的两个 32 位尺寸域组成。标志域指定应用程序的操作特征，尺寸域指示应用程序的最小及首选分区的尺寸。最小分区的尺寸是一个下限值，低于它，您的应用程序就不能工作。首选分区尺寸是您的应用程序可以最有效地运行，并且操作系统在应用程序启用前试图保证的存储器大小。在存储器数量达不到首选分区大小的情况下，只要可用的最大连续存储器块大于指定的最小存储器，您就可以将应用程序调入其中运行。

附 注

如果可用存储器空间在最小和首选分区尺寸之间，Finder将显示一个对话框，询问用户是否希望在可用的存储器中运行应用程序。如果您的应用程序没有‘SIZE’资源，它将被指派一个 512 KB 的系统设定分区尺寸。而且，正常情况下由标志域中的常数定义的所有规格的进程管理器都使用 FALSE 系统缺省值 。

当您定义 'SIZE' 资源时，您应该给它一个资源标识符值 -1。用户可以在 Finder 信息窗口中修正应用程序的首选大小。若用户没有更改分区大小，操作系统就会创建一个新的 'SIZE' 资源，其资源标识符为 0。在应用程序启用的时候，进程管理器搜索一个标识符为 0 的资源 'SIZE'；若未找到这个资源，它就使用原来标识符为 -1 的 'SIZE' 资源。同样，在用户修正资源中的其它设定状态时也要创建这种新的 'SIZE' 资源。

当创建 'SIZE' 资源时，您首先需要确定应用程序的各种操作特征。例如，当您的应用程序跑在后台时，如果它不做什么有用的工作，那么您就不应该设定 canBackground 志。同样，如果您还没有在使用全 32 位间接指针或指针作存储器地址的环境中测试过您的应用程序，您也不应该设定 is32BitCompatible 标志。

程序清单 2-4 是一个示例 'SIZE' 资源的 Rez 输入（Rez 是一个资源编译程序，可以在 MPW 环境下使用）。

程序清单 2-4 示例 'SIZE' 资源的 Rez 输入

```
resource 'SIZE' (-1) {
     reserved,                       /*reserved*/
     acceptSuspendResumeEvents,      /*accepts suspend&resume
events*/
     reserved,                       /*reserved*/
     canBackground,                  /*can use background null */
                                     /* events*/
     doesActivateOnFGSwitch,         /*activates own windows in */
                                     /* response to OS events*/
     backgroundAndForeground,        /*application has a user */
                                     /* interface*/
     dontGetFrontClicks,             /*don't return mouse events */
                                     /* in front window on resume*/
     ignoreAppDiedEvents,            /*doesn't want app-died events*/
     is32BitCompatible,              /*works with 24- or 32-bit
addr*/
     isHighLevelEventAware,          /*supports high-level events*/
     localAndRemoteHLEvents,         /*also remote high-level events*/
     isStationeryAware,              /*can use stationery documents*/
     dontUseTextEditServices,        /*can't use inline input */
                                     /* services*/
     reserved,                       /*reserved*/
     reserved,                       /*reserved*/
     reserved,                       /*reserved*/
     kPrefSize * 1024,               /*preferred memory size*/
     kMinSize * 1024                 /*minimum memory size*/
};
```

程序清单 2-4 中的 'SIZE' 资源规范表明，应用程序可以接收挂起和恢复执行事件；跑在后台时可利用空事件处理任务；作为对操作系统事件的响应，必要时可以激活和失活任何窗口；既能在前台也能在后台执行；以及不希望接收与因用户点按应用程序前面的窗口造成的恢复执行事件有关的任何鼠标事件。它也表明应用程序不希望接收应用程序“死亡”事件；能够以 24 位或 32 位寻址方式工作；接收本地和网络高级事件；能够处理固定样板文稿以及不使用 TextEdit 在线输入服务。在这个实例中，Rez 输入文件必须定义常数 kPrefSize 和 kMinSize 的值；例如若 kPrefSize 设定为 50，首选分区大小就是 50 KB。

您指定的作为应用程序首选和最小存储器大小的数字取决于您的应用程序所需要的专用存储器的大小。您的应用程序的存储器要求取决于它的静态堆、动态堆、A5 区域和堆栈的大小（请参阅《Inside Macintosh : Memory》一书的“存储器管理器介绍”一章中关于应用程序存储器分区的详细说明)。静态堆大小包括应用程序执行过程中总是存在的对象。例如执行代码段、窗口记录的工具箱数据结构等等。

动态堆要求来自各个创建在每个文稿中的对象(它的大小可以随文稿本身的不同按比例地更改）和特殊命令或函数要求的对象。

A5 区域的大小取决于全局数据的量，以及您的应用程序所包含的跳转表个数。

堆栈包含变量、返回地址和临时信息。应用程序堆栈的大小随电脑类型的不同而不同，所以，您应该根据具体的电脑上要求的堆栈大小（例如，Macintosh Plus 所要求的堆栈大小为 8 千字节）决定您的堆栈大小。进程管理器可以通过自动调整您要求的存储器容量来对不同电脑上不同的堆栈大小进行补偿。例如，若您要求 512 KB，在系统设定堆栈较大的电脑上将分配较大的堆栈空间（大约 16 千字节)。

遗憾的是，很难较可靠地预期所有这些条件。您应该能够对堆栈大小、静态堆大小、A5 区域和跳转表作出合理的估计。此外，您可以使用 MacsBug 堆探索命令这类工具来帮助您根据经验确定应用程序要求的动态存储器到小。

请参阅 2-115 页的“ 'SIZE' 资源”中更多的有关 'SIZE' 资源的每个域和标志意义的信息。

处理低级事件

低级事件包括硬件相关事件(它们储存在操作系统事件队列中)和窗口管理器生成的激活和更新事件。当您的应用程序接收一个低级事件时，需要确定事件的类型并适当地作出响应。以下各节将讨论如何对鼠标事件、键盘事件（包括某些特殊的键盘事件，例如用户同时按下 Command 键和“ · ”键)、更新事件、激活事件、磁盘插入事件及空事件做出响应。

响应鼠标事件

只要用户按下或释放鼠标按钮，操作系统事件管理器就在操作系统事件队列中记录了这个动作。这些动作作为按下鼠标和释放鼠标事件储存在事件队列中。您的应用程序可以通过调用 WaitNextEvent 函数检索这些事件。

鼠标移动事件不储存在事件队列中。鼠标驱动程序自动跟踪鼠标并随用户移动鼠标显示光标所在位置。因此，在用户只是移动鼠标时，操作系统事件管理器不对此进行报告。

但是，若用户将光标移出了您在 WaitNextEvent 函数中指定的区域，您可以要求事件管理器报告鼠标移动事件。例如，您的应用程序可以使用鼠标移动事件来改变光标的形状，当用户将光标从窗口的文字区移至滚动杆时，光标从 I 型符变成箭头形状。

本节的其余部分介绍了您的应用程序如何响应按下鼠标或释放鼠标事件。请参阅 2-62 页开始的"响应鼠标移动事件"中对鼠标移动事件的专门说明。

用户希望按下鼠标按钮这个动作是和应用程序中的特定动作是相关的。您的应用程序应该为用户提供反馈信息或执行响应用户的动作。例如，当光标在菜单栏上时，用户按下鼠标按钮，您的应用程序应该使用菜单管理器函数 MenuSelect 允许用户能够选择一个菜单命令。

您的应用程序可以接收并响应按下鼠标和释放鼠标事件。大多数应用程序都响应鼠标按下事件，并使用各种管理器的例程（例如 MenuSelect，DragWindow，TEClick、TrackBox，TrackGoAway 及 TrackControl）来处理释放鼠标事件。如果事件适合您的应用程序，您也可以提供代码段来响应释放鼠标事件。例如，当您的应用程序执行它自己的文字编辑函数时，您可能会让用户通过拖拽鼠标来选取文字的一些行，并使用释放鼠标事件表示结束选取。

在 System 7 中，只有当您的应用程序跑在前台处理时，而且是用户点按菜单栏、点按属于您的应用程序创建的窗口，或者点按一个在您的应用程序分区中启动的桌面附件的窗口时，才接收释放鼠标事件。若用户点按一个属于其它应用程序的窗口，事件管理器会向您的应用程序发送一个挂起事件，然后执行一个主切换，将应用程序中切换到前台去。

当您的应用程序接收一个鼠标按下事件时，您需要首先确定光标在按下鼠标时的位置（鼠标位置）并适当地做出响应。您可以使用窗口管理器函数 FindWindow 来查找鼠标是点按在应用程序的哪一个窗口(如果有的话)，如果需要还可找出是在窗口的哪一部分按下的。FindWindow 函数还报告鼠标位置是否点按在菜单栏上，或者在某些情况下，是否点按在一个属于桌面附件的窗口中（若该桌面附件是在您应用程序的（存储器）分区中启动的)。

鼠标事件记录的 what 域包含 mouseDown 或 mouseUp 常数，它们可以用来报告鼠标钮是按下还是释放。message 域未定义。when 域包含了系统自上次启动以来的时间值（以 Tick 为单位)。您可以使用 when 域来比较连续鼠标事件的时间间隔；例如，您可以使用这个信息帮助测试鼠标的双击动作。

事件记录的where域包含鼠标按钮按下或释放时的光标位置。您可以将此位置传送到FindWindow函数FindWindow函数将给定的鼠标位置映射到屏幕的特定区域。

modifiers 域包含有关修正键在鼠标按钮按下或释放时的状态。您的应用程序可以根据修正键的状态执行不同的动作。例如，当 Shift 键在按下鼠标时按下，您的应用程序可能会让用户一次选取多个对象。

程序清单 2-5 给出了处理鼠标按下事件的代码段。DoMouseDown 过程是一个应用程序定义的过程，它被 DoEvent 过程调用。(2-26 页中的程序清单 2-3 给出了 DoEvent 过程)。

程序清单 2-5 处理鼠标按下事件

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
    part:         Integer;
    thisWindow:   WindowPtr;
BEGIN
    {map location of the cursor (at the time of mouse-down event) }
    { to general areas of the screen}
    part := FindWindow(event.where, thisWindow);
    CASE part OF  {take action based on the mouse location}
    inMenuBar: {mouse down in menu bar, respond appropriately}
        BEGIN
            {first adjust marks and enabled state of menu items}
            MyAdjustMenus;
            {let user choose a menu command}
            DoMenuCommand(MenuSelect(event.where));
        END;
  inSysWindow: {cursor in a window belonging to a desk accessory}
        SystemClick(event, thisWindow);
```

```
inContent:     {mouse down occurred in the content area of }
               { one of your application's windows}
  IF thisWindow <> FrontWindow THEN
  BEGIN   {mouse down occurred in a window other than the front }
          { window--make the window clicked in the front window, }
          { unless the front window is movable modal}
     IF MyIsMovableModal(FrontWindow) THEN
          SysBeep(30)
     ELSE
          SelectWindow(thisWindow);
  END
  ELSE    {mouse down was in the content area of front window}
     DoContentClick(thisWindow, event);
inDrag:                  {handle mouse down in drag area}
     IF (thisWindow <> FrontWindow) AND
        (MyIsMovableModal(FrontWindow))
     THEN
         SysBeep(30)
     ELSE
         DragWindow(thisWindow, event.where,
GetGrayRgn^^.rgnBBox);
     inGrow:             {handle mouse down in grow region}
         DoGrowWindow(thisWindow, event);
     inGoAway:           {handle mouse down in go-away region}
         IF TrackGoAway(thisWindow, event.where) THEN
           DoCloseCmd;
     inZoomIn, inZoomOut:{handle mouse down in zoom box region}
         IF TrackBox(thisWindow, event.where, part) THEN
           DoZoomWindow(thisWindow, part);
     END; {end of CASE}
END; {of DoMouseDown}
```

当您的应用程序检索到鼠标驱动事件时，根据该事件的 where 域信息，调用窗口管理器 FindWindow 函数，将光标位置射到屏幕的特定区域。FindWindow 函数返回一个值这个值表示鼠标位置是在菜单栏、在您的应用程序的一个窗口上，或者可能在桌面附件窗口上。如果鼠标位置在应用程序窗口中，则函数结果表示鼠标位于该窗口的哪一部分。您可以依照下列常数来测试 FindWindow 的函数结果，以确定在鼠标驱动事件执行期间的鼠标置：

```
CONST   inDesk          = 0; {none of the following}
        inMenuBar       = 1; {in the menu bar}
        inSysWindow     = 2; {in a desk accessory window}
```

```
inContent   = 3;    {anywhere in content region except the }
                    { grow region if the window is active, }
                    { anywhere in content region including the }
                    { grow region if the window is inactive}
inDrag      = 4;    {in drag (title bar) region}
inGrow      = 5;    {in grow region (active window only)}
inGoAway    = 6;    {in go-away region (active window only)}
inZoomIn    = 7;    {in zoom-in region (active window only)}
inZoomOut   = 8;    {in zoom-out region (active window only)}
```

如果鼠标点按位置不在菜单栏、桌面附件窗口、或您的应用程序的某个窗口上FindWindow 函数返回值为一个 inDesk 常数。例如，如果光标位于窗口框架内，但又不在该窗口的拖拽区域或关闭（go-away）区域上，则FindWindow函数报告此常数；您的应用程序极少会接收到inDesk常数。

如果FindWindow返回inMenuBar常数，则表明点按于菜单栏内。此时，您的应用程序应首先调整其菜单。应用程序定义的MyAdjustMenus过程，可以根据当前活动窗口的内容调整其菜单，如启用和停用某一菜单项目以及设定标记等。例如，如果当前活动窗口是一个包含编辑排版信息的文稿窗口，您的应用程序应在编辑菜单中启用"剪"和 "复制"命令，在字体大小字形菜单中的相应项目上添加标记、并适当调整其它菜单命令项目。在调整了您的应用程序的菜单后，调用菜单管理器函数MenuSelect，将鼠标位置传递给它，以便用户选择菜单命令。MenuSelect 函数处理所有用户和机器的交互作用，直到用户释放鼠标按钮为止。MenuSelect函数返回一个长整数，以标示用户选择的菜单。如2 – 34 页上的程序清单2 – 5 所示，例程DoMouseDown调用应用程序定义的例程DoMenuCommand来执行用户选择的菜单命令。请参阅本书"菜单管理器"一章中关于MyAdjustMenus和DoMenuCommand例程代码段的列表和响应特定菜单命令的详细信息。

在System 7中，FindWindow函数很少返回inSysWindow常数。只有在应用程序分区启用的桌面附件窗口发生鼠标按下事件时，FindWindow函数才会返回此常数。通常，如果用户点按一个桌面附件窗口，事件管理器就会向您的应用程序发送一个挂起事件并将桌面附件切换到前台。自此，鼠标按下事件和其它事件均由桌面附件来处理，直到用户再次点按您的应用程序的一个窗口为止。

如果 FindWindow 返回了 inSysWindow 常数，则表明鼠标点按于您的应用程序分区启用的桌面附件所属的窗口中。在这种情况下，您的应用程序应调用SystemClick 过程。SystemClick 过程会适时地将事件发送给桌面附件。如果鼠标按钮被按下时，光标位于桌面附件窗口的内容区域且该窗口呈非现用状态，那么SystemClick就会将该窗口激活，使之成为当前活动窗口。此过程首先向您的应用程序发送一个激活事件停用该应用程序的前端窗口，然后向桌面附件发送一个事件激活其窗口。

如果给定的鼠标位置在您的应用程序的当前活动窗口中，FindWindow 可返回常数 inContent、inDrag、inGrow、inGoAway、inZoomIn 或 inZoomOut 中的任何一个。如果光标在内容区域中，您的应用程序应执行适用于您的应用程序的某个动作。请注意，滚动杆是内容区域的一部分。多数情况下，如果光标位于内容区域中，您的应用程序首先需要确定鼠标是否位于滚动杆或其它控制部件中，然后适当地进行响应。DoMouseDown 过程调用应用程序定义的过程 DoContentClick 来处理当前活动窗口内容区域中的鼠标按下事件。如果您的应用程序需要确定鼠标按下事件是否引起前台交换(并且在您的应用程序的 'SIZE' 资源中已设定 getFrontClicks 标志)，您的 DoContentClick 过程可以测试事件记录 Modify 域中的位 0（通常您的应用程序不测试此状态）。请参阅本书“控制管理器”一章中关于 DoContentClick 过程的实例和关于在您的应用程序窗口中实现控制的详细说明。

如果鼠标位于一个现用应用程序窗口的任何其它特定的区域中，您的应用程序应执行针对该区域的动作。例如，如果光标位于拖拽区域中，您的应用程序应调用窗口管理器的 DragWindow 过程，以便用户能够将窗口拖拽到新的位置。

如果光标位于一个非现用应用程序窗口，FindWindow 可返回 inContent 或 inDrag 常数，但不能将窗口的其它各个区域区分开来。在此情况下，如果 FindWindow 报告 inContent 常数，您的应用程序应利用 SelectWindow 过程将这个非当前活动窗口调到前台来（除非该当前活动窗口是可移动定模式对话框）。如果当前活动窗口是可移动定模式对话框，当用户点按其它窗口时，您应调用 SysBeep 过程发出系统示警声，而不是激活用所选择的窗口。同样，如果您的应用程序在一个非当前活动窗口中将第一个鼠标点按事件解释为一个激活窗口的请求并执行一个动作，那么您可以再次处理该事件。但请注意，多数用户希望第一次点按就可以将非当前活动窗口激活，而不必执行任何附加动作。对于一个非现用应用程序窗口，如果 FindWindow 的返回值为 inDrag，您的应用程序应调用 DragWindow 过程，让用户将该窗口拖拽到一个新的位置（除非该当前活动窗口是一个可移动定模式对话框，在此情况下，您的应用程序只应发出系统示警声）。

如果您使用 TextEdit 来处理文本编辑并且调用 TEClick 例程（TEClick 就会自动而适当地解鼠标双击事件，包括允许用户通过双击选择一个英文单词）。您的应用程序必须提供可在所有其它上下文中使用上述双击方式的途径。

通过比较鼠标按下事件和释放鼠标事件的时间和位置，您就可以检测鼠标双击事件 GetDblTime 函数可以返回设定的 Tick 差，Tick 差是指释放鼠标事件和鼠标按下事件（两个鼠标事件可认为是一个双击事件）之间的时间差。

如果符合以下两个条件，您就应将两个鼠标事件解释为一个鼠标双击事件:

- 释放鼠标事件和鼠标按下事件的时间Tick 差小于等于GetDblTime函数返回的值
- 由释放鼠标事件前后的两个鼠标按下事件的位置十分接近。如何确定这个值，取决于您的应用程序和发生鼠标按下事件的上下文。例如，在字处理应用程序中，如果两个鼠标位置均映射在同一个字符上，您可能会将这两个鼠标按下事件视为双击事件; 而在图形应用程序中，如果两个鼠标位置的水平和垂直位差之和小于5个像素，您就可以将它视为双击事件

另外，事件管理器还提供可以给出鼠标信息的其它例程。您可以利用GetMouse过程查找当前的鼠标位置。您也可以调用Button、StillDown、和WaitMouseUp函数来确定鼠标按钮的当前状态。请参阅第2 － 108页的“读取鼠标”一章中有关这些例程的详细信息。

响应键盘事件

您的应用程序可以接收键盘事件，以便通知您什么时候用户按下或释放一个键或继续按住一个键。当用户按下一个键时，操作系统事件管理器便将键按下事件储存在操作系统事件队列中。您的应用程序从该队列中检索到这一事件; 确定按下了哪个键确定事件发生时按下的修正键（如果有的话）; 同时进行适当的响应。通常，您的应用程序会提供反馈信息，即将代表按键动作生成的字符显示在屏幕上。

当用户将一个键按住一段时间后，事件管理器生成自动键事件。当原始的键按下事件经过一段初始延迟达到某个延迟点（自动键阈值）时，事件管理器就会生成一个自动键事件。每当最后一个自动键事件达到某个重复时间间隔（自动键速率）且原始键仍在被按下时，事件管理器将生成后续的自动键事件。用户可以使用键盘控制板设定初始延迟及其重复速率。自动键阈值的系统设定值是16Tick，自动键速率的系统设定值是4Tick。自动键阈值和自动键速率的当前值储存在系统的全局变量KeyThresh和KeyRepThresh中。

当用户按下或释放一个键时，除了得到键盘事件外，您还可以通过调用GetKeys过程直接读取键盘（和数字小键盘sleep 参数指定了在没有事件等待您的应用程序处理的情况下，该程序同意释放CPU的时间（以Tick 为单位)。当时间到了或有了等待您的应用程序处理的事件，进程管理器会安排您的应用程序来处理。一般来说，sleep 参数应该指定为一个大于 0 的值，使其它应用程序可以在需要时占用CPU时间。例如，当用户在编辑文字，而您的应用程序需要周期性地闪烁插入光标，或使用TextEdit 来闪烁插入光标时，您的应用程序指定的值不应该大于 GetCaretTime 函数返回的值。

用 mouseRgn 参数指定一个屏幕区域，该区域采用全局坐标系，一旦鼠标移出该区域，事件管理器生成一个鼠标移动事件。当您的应用程序接收到一个鼠标移动事件时，它应重新计算 mouseRgn 参数; 若您传送的是一个空区域或 NIL 区域间接指针，事件管理器则不返回鼠标移动事件。mouseRgn参数可作为更改光标形状的简单方法，例如，当用户将光标从窗口的内容区移到滚动杆时光标形状即会发生变化。请参阅 2-62 页的“响应鼠标移动事件”中有关如何设定和更改mouseRgn参数的内容。

亦请注意，某些键盘的物理布局有所不同或包含附加键，例如功能键。如果您的应用程序支持功能键或其它特殊键，那么当用户按下这些键中的一个键后需要您确定应采取的动作时，则应遵循《macintosh Huamn Interface Guidelines》的准则。

某些按键组合是由事件管理器来处理的，而且不返回到您的应用程序中。比如Command + Shift 键 + 数字的组合就会产生特殊作用。苹果公司提供三个标准的Command - Shift - 数字键序列。标准的Command - Shift - 数字键的组合：1用于推出内部驱动器中的磁盘；2用于推出第二内部驱动器中的磁盘，如果电脑仅有一个内部驱动器，则推出外部驱动器中的磁盘3用于屏幕拍照并将其作为TeachText文稿储存在启动卷中.

对应于Command - Shift - 数字键序列的动作是作为一个例程来执行的，该例程不用参数并储存在一个带有资源标识符（与激活资源的数字对应）的'FKEY'资源中。菜（Apple）将带有资源标识符 1 至 4 的'FKEY'资源留作自用资源；如果您要提供一个'FKEY'资源，请使用资源标识符 5和9之间的资源。

将系统全局变量ScrDmpEnb（1个字节）设为0，您就可以使事件管理器对编号3 - 9的Command - Shift - 数字键序列的处理失效。但是，多数情况下，您不应中止事件管理器对这些事件的处理。

用于键盘相关事件的事件记录中的What域包含键按下和键释放常数，分别表示该键被按下或释放；或者包含自动键常数，表示该键被持续按下。

事件管理器为您的应用程序设定了系统事件类型屏蔽参数，以便接受除键释放事件以外的所有事件。多数应用程序都忽略键释放事件。如果您的应用程序需要接收键释放事件，您也可以利用操作系统事件管理器的SetEventMask过程来更改您的应用程序的系统事件类型屏蔽参数。

在低位字中，Message域中包含与用户按下的键相对应的字符代码和模拟键代码。

模拟键代码代表用户按下或释放的键；对于特殊键盘上的特定物理键，值总是相同的。例如，在苹果键盘II，ISO布局的键盘上，Tab键右边第5个键的模拟键代码（标为“T”的键）总是$11，不论按下哪个修正键都一样。

若要确定相应的特定物理键的模拟键代码，系统软件需使用硬件特定键映象（'KMAP'）资源指定特殊键盘的模拟键代码。在确定了用户按下的键的模拟键代码之后，系统软件使用语系特定键盘布局（'KCHR'）资源将模拟键代码映射到特定字符代码。所有给定的语系系统均有一个或多个'KCHR'资源。例如，某种特殊电脑可能除标准的美国'KCHR'资源外，还包含法国'KCHR'资源。在这种情况下，当前'KCHR'资源可决定是否将模拟键代码映射到法国或美国字符集。

字符代码代表一个特殊字符。这种字符代码是按照模拟键代码、修正键状态和当'KCHR'资源来生成的。例如，美国'KCHR'指定：对于模拟键代码 $ 2D（Shift键左边第5个键，在苹果键盘II，国内布局的键盘中称为“N”），当未按下修正键时，字符代码为 $ 6E；当该键与 Shift 键组合按下时，字符代码为 $ 4E。用于罗马语系系统的字符代码均在 ASCII（美国信息交换标准代码）扩充版中指定。

Message域中包含着有关ADB键盘的附加信息。高位字的低位字节中包含发生键盘事件的键盘 ADB 地址。图 2 － 7 是用于键盘事件的事件记录中 Message 域的结构。

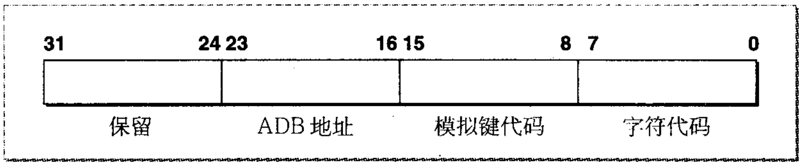


图 2-7　用于键盘事件的事件记录中的 Message 域

当您的应用程序响应键盘事件时，通常使用字符代码，而不使用模拟键代码。您可以利用这两个常数存取 message 域中的模拟键代码和字符代码：

```
CONST  charCodeMask    = $000000FF;  (字符代码存取参数)
        keyCodeMask    = $0000FF00;  (模拟键代码存取参数)
```

在 When 域中包含系统最后一次启动以来的 Tick 数。您可以使用 When 域来比较两次相继键盘事件之间的时间差。

事件记录的 Where 域包含按下或释放键时光标所在的位置。当处理键盘事件时，您一般不用考虑鼠标的位置。

modifiers域包含按下或释放键时有关修正键状态的信息。您的应用程序可以根据修正键的状态来执行不同的动作。例如，如果在键按下事件期间按下 Command 键，您的应用程序可能会执行一个与相应的菜单命令有关的动作。

系统软件可以支持数种不同类型的键盘，例如，苹果键盘II、苹果扩充键盘、或其它键盘。系统软件利用各种键盘资源和国际资源来管理不同类型的键盘。图2 － 8为系统软件如何将键映射到字符代码的图示。

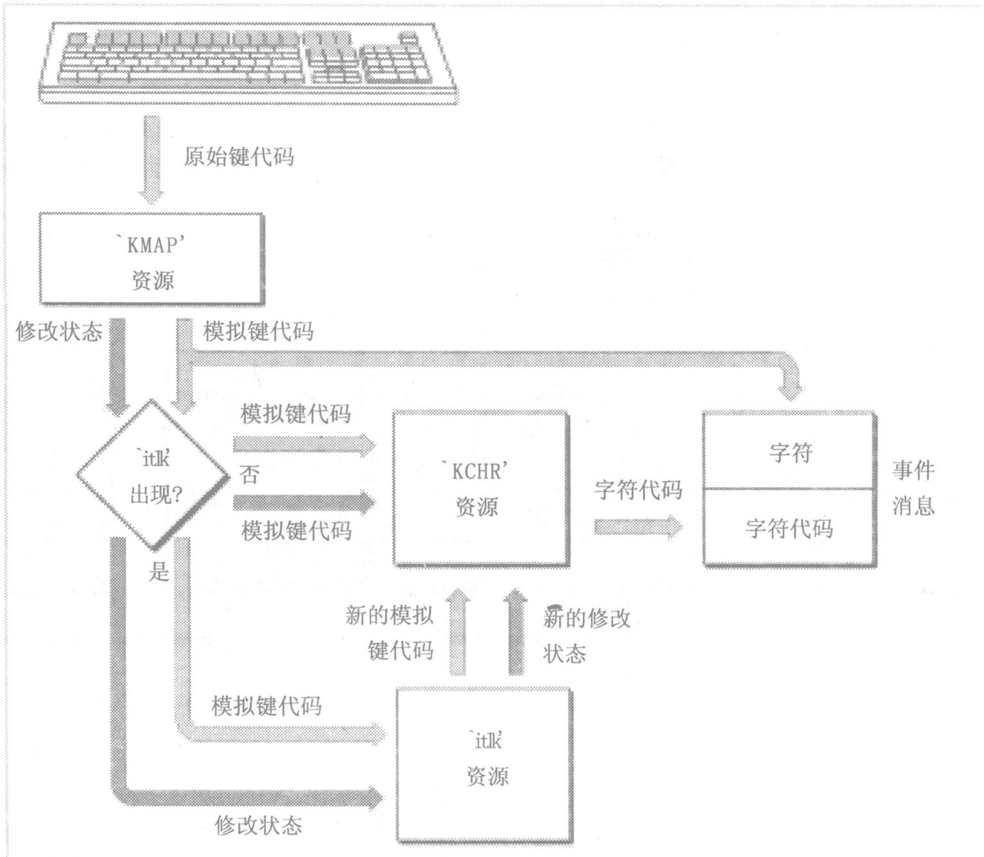


图2-8　键盘转换

当用户按下或释放键盘上的一个键时，键盘生成一个原始键代码。系统软件使用`KMAP'资源将原始键代码映射到硬件无关的模拟键代码并设定指示修正键状态的位。`KMAP'资源指定特殊键盘的物理排列并标示对应每个物理键的模拟键代码。

如果有可选的键重新映像（`itlk'）资源，那么在使用`KCHR'资源之前，系统软件将针对某些键盘上的一些组合键重新映射模拟键代码和修改状态。`itlk'资源可以重新引入硬件相关设备，原因是某些语系、语言和区域需要布局略有不同的特定键盘。如果有，那么`itlk'资源将只影响少数几个键。

当可选的`itlk'资源映射模拟键代码和修正键状态的工作完成之后，系统软件使用`KCHR'资源来产生代表按下和释放键的字符代码。`KCHR'资源指定了将修正键和模拟键代码的设定状态映射到字符代码的方法。

映射了该键之后，事件管理器返回事件记录的Message域中的模拟键代码和字符代码。

图2－9是'KMAP'资源为苹果键盘II，ISO布局的键盘指定的模拟键代码图示。如图所示的键盘是采用美国键盘布局来标示每个键的。图示中模拟键代码为十六进制代码。

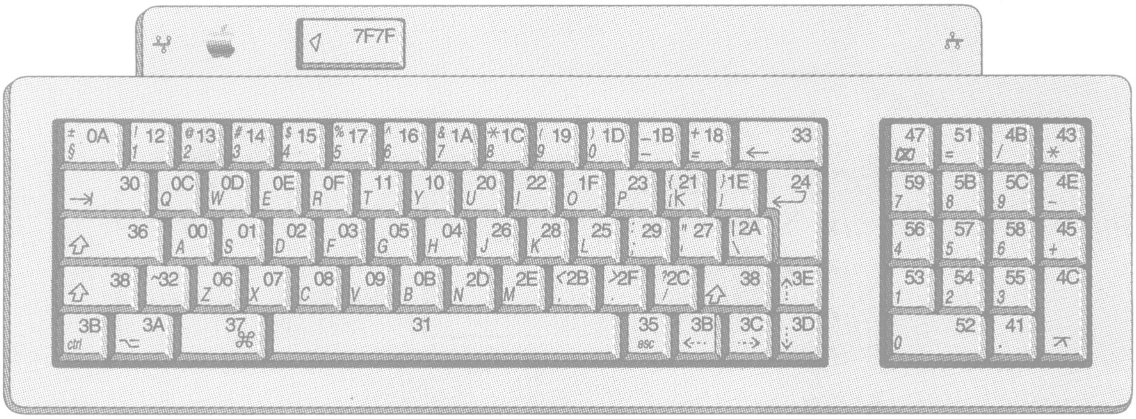


图2-9　苹果键盘II，ISO布局的模拟键代码

图2－10所示的'KMAP'资源指定的模拟键代码用于Apple扩充键盘II，其中一个使用国内布局（ANSI)，另一个使用ISO布局。如图所示的ISO键盘上的键使用法国键盘布局标示。图示中的模拟键代码为十六进制代码。

如果苹果扩充键盘II（使用美国'KCHR'资源）的用户按下标示为“C”的键且未按下修正键，则系统软件通过'KMAP'和'KCHR'资源对该键进行映射，在事件记录的message域中产生模拟键代码＄08 和字符代码＄63（字符“c”)。如果用户按下标示为“C”的键和Option键，那么，系统软件在Message域中便将其映射为模拟键代码＄08和字符代码＄8D（字符“　”)。

在另一个实例中，用户使用苹果扩充键盘II（国内布局)，如果使用美国'KCHR'资源并按下标示为“M”的键，则系统软件通过'KMAP'和'KCHR'资源对其映射，目的是要在事件记录的message域中产生模拟键代码＄2E和字符代码＄6D(字符“m”)。

如果苹果扩充键盘II（ISO)布局的用户使用法国'KCHR'资源并按下标示为“M”的键，则系统软件通过'KMAP'和'KCHR'资源对其映射，目的是要在事件记录的message域中产生模拟键代码＄29和字符代码＄6D（字符“m”)。

请参阅《Inside Macintosh文本》(*Inside Macintosh: Text*) 中关于键盘资源和语系管理器如何管理各种语系的附加信息。

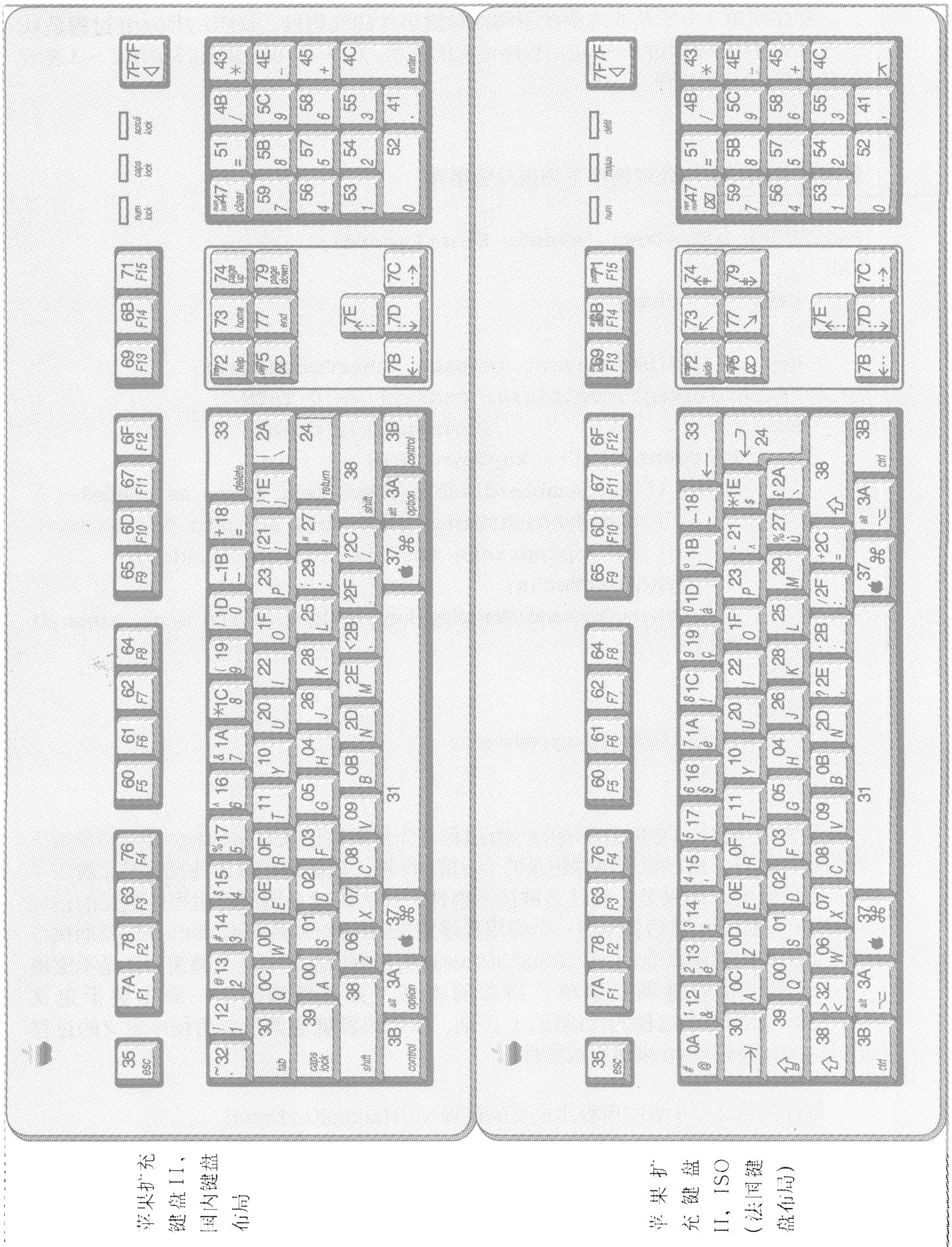


图 2-10　苹果扩充键盘II模拟键代码

程序清单2 - 6是处理键按下和自动键事件的代码段。DoKeyDown过程是从DoEvent过程中调用的一个应用程序定义的过程。(2 - 26页上的程序清单2 - 3表示的就是DoEvent过程。)

程序清单2 - 6　处理键按下和自动键事件

```
PROCEDURE DoKeyDown (event: EventRecord);
VAR
     key:      Char;
BEGIN
     key := CHR(BAnd(event.message, charCodeMask));
     IF BAnd(event.modifiers, cmdKey) <> 0 THEN
     BEGIN                     {Command key down}
          IF event.what = keyDown THEN
          BEGIN {first enable/disable/check menu items as needed-- }
                { the MyAdjustMenus procedure adjusts the menus }
                { as appropriate for the current window}
              MyAdjustMenus;
              DoMenuCommand(MenuKey(key));{handle the menu command}
          END;
     END
     ELSE
         MyHandleKeyDown(event);
END;
```

程序清单2 - 6中的DoKeyDown过程首先从事件记录的Message域中析取按下键的字符代码，然后核查事件记录的modifiers域，以确定在该事件期间是否按下了Command键。如果是，而且是键按下事件，该代码就会调用应用程序定义的过程MyAdjustMenus，然后调用另一个应用程序定义的例程DoMenuCommand，从而执行与该键相关的菜单命令。(MyAdjustMenus过程根据当前窗口是文稿窗口还是不定模式对话框来相应地调整菜单。请参阅本书“菜单管理器”一章中关于定义MyAdjustMenus过程的代码段。)否则，该代码段就会调用应用程序定义的过程MyHandleKeyDown来执行该事件。

程序清单2 - 7是应用程序定义的例程MyHandleKeyDown。

程序清单2 - 7　处理键按下事件

```
PROCEDURE MyHandleKeyDown (event: EventRecord);
VAR
    key:                    Char;
    window:                 WindowPtr;
```

```
    myData:                      MyDocRecHnd;
    te:                          TEHandle;
    windowType:                       Integer;
BEGIN
    window := FrontWindow;
   {determine the type of window--document, modeless, etc.}
    windowType := MyGetWindowType(window);
    IF windowType = kMyDocWindow THEN
    BEGIN
        key := CHR(BAnd(event.message, charCodeMask));
        IF window <> NIL THEN
        BEGIN
            IF key = char(kTab) THEN{handle special characters}
                         MyDoTab(event)
            ELSE
            BEGIN
                myData := MyDocRecHnd(GetWRefCon(window));
                te := myData^^.editRec;
                IF
                 (te^^.teLength - (te^^.selEnd - te^^.selStart) + 1
                   < kMaxTELength) THEN
                BEGIN
                    TEKey(key, te);{insert character in document}
                    MyAdjustScrollBars(window, FALSE);
                    MyAdjustTE(window);
                    myData^^.windowDirty := TRUE;
                END;
            END;
        END;
    END
    ELSE
       MyHandleKeyDownInModeless(event, windowType);
END;
```

程序清单2－7中的MyHandleKeyDown过程可以处理任何应用程序窗口中的键按下事件。对于文稿窗口，该代码段可以将用户按下的键代表的字符插入现用文稿中。它首先利用FrontWindow函数查找现用文稿，然后适当地处理该文稿窗口中的事件。例如，它将Tab键作为一个特殊字符并调用应用程序定义的例程MyDoTab来适当地处理这个文稿字符的宽度。对于输入到文稿窗口的所有其它键，该代码段首先获取与该文稿相关的编辑记录，然后使用TextEdit TEKey过程将字符插入文稿。它也可以调用另外两个应用程序定义的例程，MyAdjustScrollBars和MyAdjustTE，以便更新文稿和编辑记录。

M y H a n d l e K e y D o w n 过程调用应用程序定义的例程 MyHandleKeyDownInModeless来处理不定模式对话框中的键按下事件。请参阅本书“对话管理器”一章中关于处理对话框中事件的详细信息。

扫描取消事件

您的应用程序应允许用户使用“Command”+“·”键组合来取消冗长的操作。通过利用GetKeys过程定期检验键盘状态，您的应用程序就可以执行此取消操作，或者，您的应用程序可以扫描键盘事件的事件队列。

程序清单2-8是一个应用程序定义的函数，该函数可以扫描事件队列中发生的任何“Command”+“·”键组合事件。

程序清单2-8中的UserDidCancel函数首先显示用户是否更改了语系。应用程序保留一个全局变量gCurrentKey语系，用以跟踪该信息。应用程序也使用一个全局变量gPeriodKeyCode保留一个键代码，这个键代码可以根据当前的语系映射到“·”键。如果当前语系已被更改，UserDidCancel函数调用应用程序定义的例程MySetPeriodKeyCode，以便需要时更改gPeriodKeyCode全局变量的值。

然后，UserDidCancel函数确定A/UX是否正在运行。如果A/UX在运行，您必须调用不同的方法扫描事件队列。如果A/UX正在运行，此代码段使用一个称为MyCheckAUXEventQueue的应用程序定义的函数来搜索“Command”+“.”键组合事件。否则，该代码段就会核查What域中的键按下事件。如果找到一个键按下事件，它便接着核查message域以确定用户是否按下了“.”键，同时核查modifiers域以确定用户是否按下了Command键。如果找到“Command”+“.”键组合，则将foundEvent变量设定为TRUE并返回该值。否则，它会查看队列中的下一个输入并继续搜索该队列直至发现一个“Command”+“.”键组合事件或到达队列尾部。

程序清单2-8 扫描“Command”+“·”组合键事件

```
FUNCTION UserDidCancel: Boolean;
VAR
    foundEvent:    Boolean;
    eventQPtr:     EvQElPtr;
    eventQHdr:     QHdrPtr;
    keyCode:       LongInt;
    isCmdKey:      LongInt;
BEGIN
    foundEvent := FALSE;             {assume the event is not there}
   {Check to see if the script has changed}
    IF (gCurrentKeyScript <> GetEnvirons(smKeyScript)) THEN
       MySetPeriodKeyCode;           {set gPeriodKeyCode to match new script
```

```
   IF (GetAUXVersion > 0) THEN  {if A/UX is running use this method}
      foundEvent := MyCheckAUXEventQueue(gPeriodKeyCode, cmdKey)
   ELSE
   BEGIN                                 {scan event queue}
      eventQHdr := GetEvQHdr;            {get the event queue header}
      eventQPtr := EvQElPtr(eventQHdr^.qHead); {get first entry}
      WHILE (eventQPtr <> NIL) AND (NOT(foundEvent)) DO
      BEGIN                              {look for key-down event}
         IF (eventQPtr^.evtQWhat = keyDown) THEN {found key-down event, }
         BEGIN                           { look for Command-period}
            keyCode := BAND(eventQPtr^.evtQMessage, keyCodeMask);
            keyCode := BSR(keyCode, 8);
            isCmdKey := BAND(eventQPtr^.evtQModifiers, cmdKey);
            IF is CmdKey <> 0 THEN       {Command key was pressed}
               IF keyCode = gPeriodKeyCode THEN
                foundEvent := TRUE;      {key pressed was '.'}
         END;     {of found key-down}
         IF (NOT foundEvent) THEN         {go to next entry}
           eventQPtr := EvQElPtr(eventQPtr^.qLink);
      END;        {of while}
   END;           {of scan event queue}
   UserDidCancel := foundEvent;          {return result of search}
END;
```

响应更新事件

每当您的一个应用程序的窗口需要更新时，事件管理器就会向您的应用程序报告更新事件。一旦接收到一个更新事件，您的应用程序应更新指定窗口的内容。首先调用窗口管理器过程 BeginUpdate，绘制窗口的内容，完成对窗口内容的更新后，调用 EndUpdate 过程结束这一次更新事件的响应。

通过在窗口记录中向包含有窗口内容的图片提供一个间接指针，您的应用程序就可以使窗口管理器自动地更新窗口内容。此技术一般仅用于那些包含有不可更改或不可编辑的静态信息的窗口。例如，如果您的应用程序提供的窗口总是显示一个地球图片，您可以为该图片提供间接指针，窗口管理器则在需要时自动地更新窗口，而不向您的应用程序发送更新事件。多数情况下，您的应用程序需要自己执行更新操作。

窗口管理器为每个窗口保留了一个更新区域。窗口管理器跟踪所有需要内容刷新的窗口内容区域并把它们累加更新区域中，一旦更新区域非空时，事件管理器就向相应的应用程序报告更新事件；带有要求更新窗口内容的应用程序可根据正常的事件处理进程接收必要的更新事件。

如果多个窗口需要更新，事件管理器首先向最前端的窗口发出更新事件。这意味着窗口的更新是根据用户的期望从前端至后端的顺序进行的。

当您的一个应用程序的窗口需要更新时，窗口管理器调用该窗口的窗口定义函数，请求绘制窗口框架。窗口管理器然后为该窗口生成一个更新事件。事件管理器向您的应用程序报告用于您的应用程序窗口的所有更新事件，而您的应用程序应根据需要更新窗口内容。

为了响应更新事件，您的应用程序首先应调用 BeginUpdate 过程。BeginUpdate 过程用该窗口的可视区域和更新区域的交集暂时替换窗口图形端口的可视区域（在屏幕上可看到的窗口部分）。然后，BeginUpdate 过程清除窗口的更新区域－－防止再次报告此次发生的更新事件。

调用了 BeginUpdate 之后，您的应用程序应全部或部分地绘制窗口的内容。您可以绘制整个内容区域也可以只绘制可视区域。对于这两种情况，窗口管理器仅允许在屏幕上绘制可视区域中的内容。（因为 BeginUpdate 过程将可视区域与更新区域相交，此时的可视区域相当于旧的更新区域的任何可视部分。）

EndUpdate 过程还原窗口图形端口的正常可视区域。

图 2－11 表示应用程序如何更新其窗口。在这个实例中，窗口 1 覆盖了窗口 2 的一部分。当用户移动窗口 1，使窗口 2 更多地暴露出来时，窗口管理器请求窗口的窗口定义函数更新窗口框架、累加需要在窗口更新区域进行更新的面积。

当应用程序接收到用于该窗口的更新事件时，事件记录的 message 域中就会包含指向需要更新的窗口的指针。您的应用程序可以调用 BeginUpdate 绘制窗口内容，然后调用 EndUpdate。这样就完成了对更新事件的处理。

无论您的应用程序跑在前台或后台均可接收更新事件。在图 2－11 所示的实例中，窗口 1 和窗口 2 可以属于同一个应用程序或属于不同的应用程序。在这种情况下，事件管理器会向需要更新窗口内容的应用程序报告更新事件。

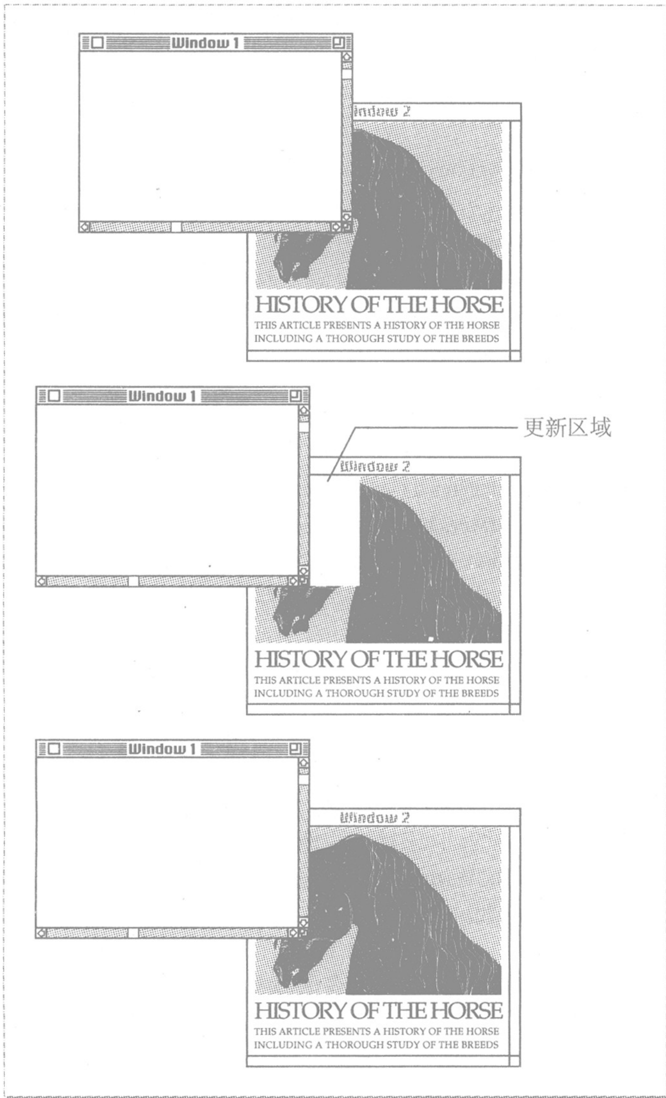


图 2-11　响应窗口更新事件

您的应用程序应该响应更新事件，或至少调用 BeginUpdate 过程对一更新事件作出响应。如果您不调用 BeginUpdate 过程，那么您的应用程序将继续接收窗口更新事件（直到更新区域变空为止）。您应该时刻保证在调用了 BeginUpdate 之后，调用 EndUpdate 来与之匹配。您可以通过调用 BeginUpdate 和 EndUpdate 过程对，告诉窗口管理器您已经更新了窗口内容，并处理了更新事件。

程序清单2－9是一个由应用程序定义的例程实例，这个例程可以响应更新事件。

程序清单2－9 响应更新事件

```
PROCEDURE DoUpdate (window: WindowPtr);
VAR
    windowType: Integer;
BEGIN
   {determine the type of window--document, modeless, etc.}
    windowType := MyGetWindowType(window);
    CASE windowType OF
        kMyDocWindow:
            BEGIN
                BeginUpdate(window);
                MyDrawWindow(window);
                EndUpdate(window);
            END;
        OTHERWISE
            DoUpdateMyDialog(window);
     END; {of CASE}
END;
```

程序清单2－9中的DoUpdate过程，首先确定窗口是一个文稿窗口还是一个不定模式对话框。MyGetWindowType函数是一个由应用程序定义的例程，当窗口是一个文稿窗口时，它返回kMyDocWindow常数，而当窗口是一个不定模式对话框时，它返回其它应用程序定义的常数。

如果窗口是一个文稿窗口，该过程则在调用BeginUpdate和EndUpdate期间，完成窗口的所有绘图工作。应用程序定义的例程MyDrawWindow执行文稿窗口内容的实际更新工作。请参阅本书“窗口管理器”一章中关于MyGetWindowType和MyDrawWindow例程代码段的内容。

如果窗口是一个不定模式对话框，该代码段就可以调用应用程序定义的例程DoUpdateMyDialog过程来更新对话框的内容。请参阅本书“对话管理器”一章中关于在对话框中处理更新事件的详细介绍。

响应激活事件

当您的应用程序的几个窗口同时打开时，您应允许用户通过点按相应的窗口从一个窗口切换到另一个窗口。每当您的应用程序接收到一个鼠标按下事件时，您应该首先借助窗口管理器函数FindWindow来判断用户是否在点按另一个窗口。如果是，那么您就可以调用窗口管理器过程SelectWindow来产生必要的激活事件。

在返回到您的应用程序之前，并在您的应用程序接收任何与这次事件相关的事件之前，SelectWindow过程将为您做一些工作，如取消停用窗口的醒目显示并醒目显示新激活的窗口。在您的应用程序下次请求事件时，事件管理器将会返回一个激活事件。

激活事件指明了所涉及的窗口，以及该窗口是被激活的还是被停用的。为完成能够使窗口变成现用或停用窗口的动作，您的应用程序应该执行该动作所需的其它动作。例如，当一个窗口变成当前活动窗口时，您的应用程序应该显示所有滚动杆并根据需要还原选定项。

对于被停用的窗口，您的应用程序一般先接收一个激活事件(带有一个表明该窗口应该被停用的标志)，随后收到的是一个针对当前活动窗口的激活事件。

激活事件并不放在操作系统事件队列中，而是被直接发送到事件管理器。

图2－12，显示了属于同一应用程序的两个文稿，其中窗口1是当前活动窗口。当用户点按窗口2时，您的应用程序会接收到一个鼠标按下事件，并可用FindWindow函数来判断鼠标位置是否在一个后台窗口。如果是，您的应用程序就应调用SelectWindow过程。SelectWindow过程将取消窗口1的醒目显示，醒目显示窗口2，并为这两个窗口生成激活事件。事件管理器每次向您的应用程序报告一个这样的激活事件；在本实例中，第一个激活事件指明窗口1应该被停用。您的应用程序应该隐藏其滚动杆，并根据需要取消所有选定项的醒目显示。

下一个激活事件表明窗口2被激活。您的应用程序应该根据需要显示滚动杆，并还原所有选定项。如果激活后窗口需要更新，事件管理器将向您的应用程序发送一个更新事件，以便您的应用程序能更新窗口内容。

在响应挂起和恢复执行事件时，您的应用程序也需要激活和停用窗口。如果您在应用程序的‘SIZE’资源中设定了acceptSuspendResumeEvents标志和doesActivateOnFGSwitch标志，那么您的应用程序将负责激活或停用其窗口，以响应处理挂起和恢复执行事件。如果您设定了acceptSuspendResumeEvents标志，而没有设定doesActivateOnFGSwitch标志，您的应用程序在收到一个挂起或恢复执行事件之后会紧接着收到一个激活事件。多数情况下，您应该在应用程序的‘SIZE’资源中设定acceptSuspendResumeEvents和doesActivateOnFGSwitch两个标志。

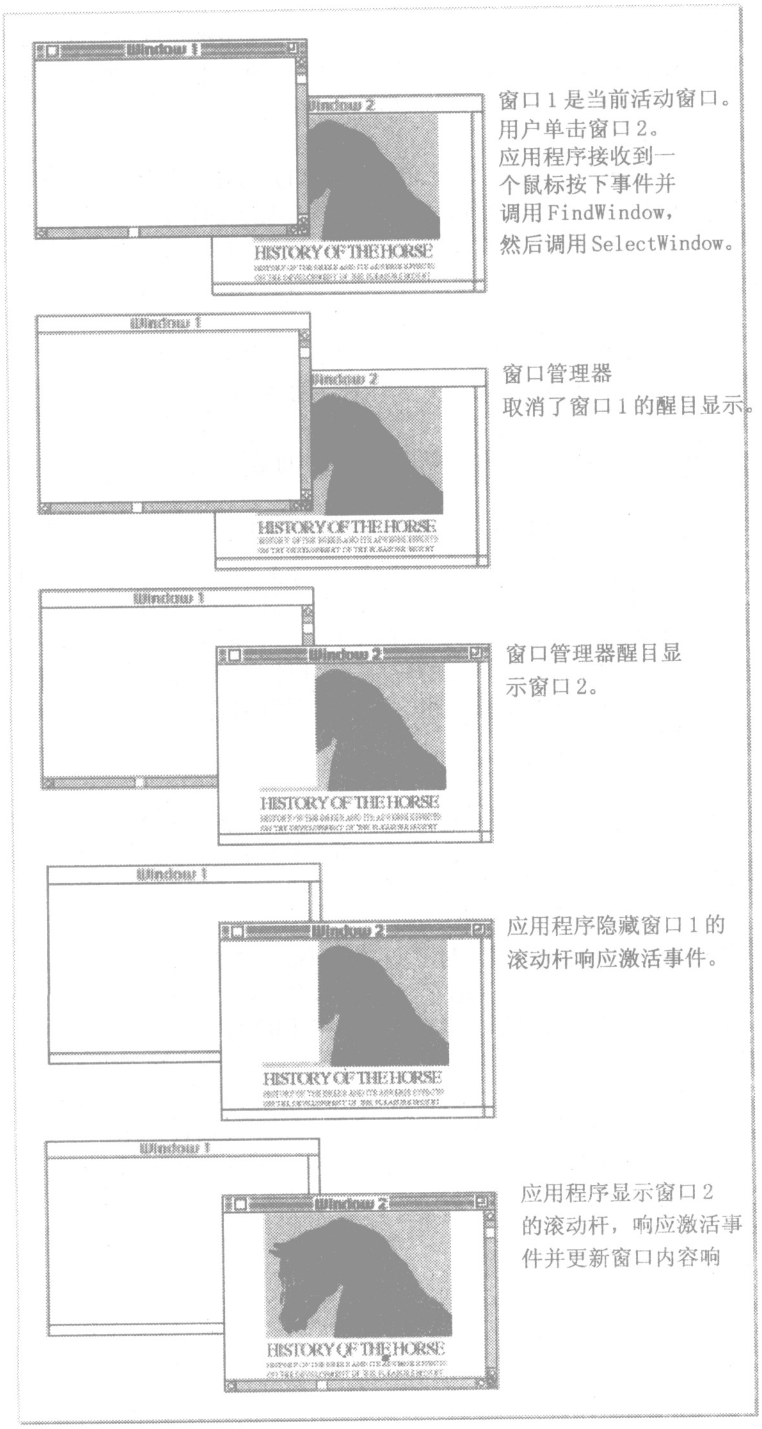


图2-12 响应窗口激活事件

一个激活事件的事件记录中的what 域，包含activateEvt 常数。message域包含一个被激活或停用窗口的指针。modifiers域包含关于激活事件的附加信息，以及该事件发送时修正键状态的信息。您的应用程序可以检查该事件记录modifiers域的位0，以判断窗口是否应该被激活或停用。如果窗口应该被激活，那么modifiers域的位0就是1；如果窗口应该被停用，则为0。您可以使用activeFlag常数来测试modifiers域位0的状态。

事件记录的when域，包含自系统上次启动以来的Tick数。事件记录的where域，包含光标在激活事件发生时的位置。

在收到一个表示窗口被停用的激活事件之后，您的应用程序应该隐藏所有滚动杆，并根据需要取消所有选定项的醒目显示。

在收到一个表示窗口正在变成当前活动窗口的激活事件之后，您的应用程序应该显示所有滚动杆，醒目显示所有选定项，或者把窗口还原到其上次被激活时的状态。例如，您的应用程序应该将插入点还原至上次位置，并将文稿卷到上次用户离开时文稿所处的位置。另外，您的应用程序还应为新的当前活动窗口正确调整其菜单－－根据当前活动窗口的状态调整菜单的标记和启用状态。

程序清单2－10是一个应用程序定义的用来响应活跃事件的过程

程序清单2－10 响应活跃事件

```
PROCEDURE DoActivate (window: windowPtr; activate: Boolean;
                      event: EventRecord);
VAR
    growRect:     Rect;          {window's grow rectangle}
    myData:       MyDocRecHnd;   {windwos's document recotd}
    windowType:   Integer;
BEGIN
    {determine the type of window--document, modeless, etc.}
    windowType := MyGetWindowType(window);
    CASE windowType OF
    kMyDocWindow:
    BEGIN
        myData := MyDocRecHnd(GetWRefCon(window));
        HLock(Handle(myData));
        WITH myData^^ DO
        IF activate THEN         {window is being activated}
```

```
    BEGIN
        {restore any selections or display caret}
        MyRestoreSelection(window);
        {adjust menus as appropriate for this document window}
        MyAdjustMenus;
        {activate any scroll bars}
        vScrollBar^^.contrlVis := kControlVisible;
        hScrollBar^^.contrlVis := kControlVisible;
        {invalidate area of scroll bars to force update}
        InvalRect(vScrollBar^^.contrlRect);
        InvalRect(hScrollBar^^.contrlRect);
        {invalidate area of size box, if any}
        growRect := window^.portRect;
        WITH growRect DO
        BEGIN
           top := bottom - kScrollbarAdjust;
           left := right - kScrollbarAdjust;
        END; {end of WITH growRect statement}
        InvalRect(growRect);
    END
    ELSE                  {window is being deactivated}
    BEGIN
       {unhighlight selection (if any) or hide the caret}
        MyHideSelection;
        HideControl(vScrollBar);    {hide any scroll bars}
        HideControl(hScrollBar);
        DrawGrowIcon(window);       {change size box immediately}
    END;
    HUnLock(Handle(myData));
  END; {end of kMyDocWindow}

  kMyGlobalChangesID:{this window is a modeless dialog box }
                     {for this app's Global Changes command}
     MyDoActivateGlobalChangesDialog(window, event);
  {handle other modeless dialog boxes as appropriate}
  END; {of CASE}
END;
```

程序清单2－10使用应用程序定义的函数MyGetWindowType来确定激活事件涉及的窗口类型。如果窗口是一个文稿窗口，DoActivate过程则dia3用GetWRefCon函数来获取窗口的文稿记录间接指针。(DoActivate过程和其它应用程序定义的例程在文稿记录中保留了与一个窗口相关联的文稿信息;应用程序储存了一个文稿间接指针，当应用程序创建一个新窗口时它就把该间接指针作为窗口的参考常数值。请参阅本书“窗口管理器”一章中关于定义一个文稿记录的介绍。)

如果文稿窗口应被激活，代码段将调用一个应用程序定义的例程 MyRestoreSelection。您的应用程序应该相应地还原选定项或显示插入光标所在位置。例如，如果您的应用程序使用 TextEdit 在窗口的内容区域显示文字，您可以调用 TextEdit 过程 TEActivate 在插入点还原选定项或显示一个插入记号。然后，DoActivate 过程调用另外一个应用程序定义的过程 MyAdjustMenus 来为文稿窗口调整菜单（请参阅“菜单管理器”中关于 MyAdiustMenus 过程的列表）。在还原了选定项并调整好菜单后，代码段就会显示被激活窗口的滚动杆和尺寸格。这项工作是这样完成的：先使滚动杆和尺寸格区域失效，然后将这些区域面积累加起来，成为更新区域。这样即可生成一个更新事件。应用程序刷新屏幕以正确响应更新事件。

如果文稿窗口将被停用，程序清单 2 - 10 中的代码段便调用应用程序定义的过程 MyHideSelection 取消选定项的醒目显示并隐藏插入雕刻的文字。然后，代码段隐藏停用窗口的滚动杆和尺寸格。

如果与激活事件相关的窗口是一个不定模式对话框，例如一个全局更改（Global Changes）不定模式对话框，则需要 DoActivate 过程在必要时调用一个应用程序定义的过程来激活或停用对话框。请参阅本书“对话管理器”一章中关于在不定模式对话框中处理激活事件的介绍。

响应磁盘插入事件

当您的应用程序使用标准文件软件包来让用户选择打开哪个文件或文件存放的位置时，标准文件软件包在与用户对话的同时响应您的应用程序的磁盘插入事件。在大多数情况下，如果您的应用程序接收到一个意外的磁盘插入事件，它能简单核查以显示磁盘是否安装成功，如果磁盘没有安装成功，它将使用磁盘初始化管理器函数 DIBadMount 来通知用户。

当用户插入磁盘时，操作系统将尽力调用文件管理函数 PBMountVol 在磁盘上安装卷。如果卷安装成功，一个代表磁盘的图符便会出现在桌面上。于是，操作系统事件管理器便生成一个磁盘插入事件。如果用户正在与一个标准文件对话框进行对话，标准文件软件包将截取该磁盘插入事件并处理它。否则该事件将被搁置于事件队列中等待您的应用程序检索。如果一个桌面附件跑在前端，桌面管理器也同样截取并处理磁盘插入事件。

通常您的应用程序应该处理而不是屏蔽磁盘插入事件。用户可能会随时插入一个磁盘，并期望磁盘在没有被初始化或被损坏时得到报警。如果您的应用程序收到一个磁盘插入事件并且成功地安装了卷，您的应用程序通常不需要采取任何更进一步的动作。但是，如果磁盘安装卷没有成功，您的应用程序就应给用户提供机会来初始化或推出没被初始化或被损坏的磁盘。

如果您屏蔽了磁盘插入事件，则该事件将留在操作系统事件队列中直到您的应用程序调用标准文件软件包或一个处理磁盘插入事件的应用程序变成前台进程为止。这种情形可能会使用户迷惑不解，所以您的应用程序应该在磁盘插入事件发生时处理它们。

如果卷已被成功地安装而您的应用程序既没有使用标准文件软件包又没有告知用户插入磁盘，那么您可以选择任意一种适合您的应用程序的方法响应磁盘插入事件。

对话管理器过程ModalDialog可以屏蔽磁盘插入事件。(标准的文件软件包为了接收磁盘插入事件而更改了掩码。)如果您的应用程序的一个定模式对话框需要响应磁盘插入事件，您可以在该事件过滤函数（这是您为ModalDialog提供的一个参数）内更改该事件类型屏蔽参数。否则，您的应用程序会在用户取消该定模式对话框后响应磁盘插入事件。

事件记录的What域中有diskEvt常数，它表示一个磁盘插入事件。message域的低位字中有驱动器编号，而高位字中有来自PBMountVol函数的结果代码。您的应用程序可以通过检验高位字来判断安装卷的尝试是否成功。如果卷没有安装成功，您的应用程序可以使用磁盘初始化管理器函数DIBadMount通知用户。如果卷安装成功，您的应用程序可以使用低位字中返回的驱动器编号存取该磁盘。

程序清单2－11显示了一个处理磁盘插入事件的过程。如果磁盘没有安装成功，该过程需使用DIBadMount函数通知用户。否则它将不采取任何动作。请参阅《Inside Macintosh: Files》的“磁盘初始化管理器”一章，以便了解磁盘初始化管理器提供的例程的有关说明。

程序清单2－11 响应磁盘插入事件

```
PROCEDURE DoDiskEvent (event: EventRecord);
VAR
    thisPoint:  Point;
    myErr:        OSErr;
BEGIN
     IF HiWord(event.message) <> noErr THEN
     BEGIN              {attempt to mount was unsuccessful}
         DILoad;        {load Disk Initialization Manager}
         SetPt(thisPoint, 120, 120);
                        {notify the user}
         myErr := DIBadMount(thisPoint, event.message);
         DIUnload;      {unload Disk Initialization Manager}
     END
```

```
    ELSE                        {attempt to mount was successful}
         ;     {record the drive number or do other processing}
END;
```

响应空事件

当事件管理器没有其他事件需要报告时，它返回一个空事件。WaitNextEvent 函数通过返回一 FALSE 函数结果报告空事件，并将返回事件记录的 what 域设定为 nullEvt。(EventAvail 和 GetNextEvent 函数也以这种方式返回空事件。)

当您的应用程序接收到一个空事件时，它可以执行空闲处理。您的应用程序应以最小量的处理工作响应空事件，以使其它过程能充分利用 CPU，同时也可以使前台进程（或者是您的应用程序，如果它在前台）能够迅速响应用户。

例如，如果您的应用程序接收到一个空事件而这个应用程序又是在前台，那么它就可以在当前活动窗口中闪烁插入记号。

如果您的应用程序在后台接收一个空事件，它就可以在后台执行任务或做其它处理。但您的应用程序不应执行可能降低前台进程响应的任何任务。如果您的应用程序在后台，同样也不应与用户对话。

如果您不希望您的应用程序在后台时接收空事件，那么就请在您的应用程序的‘SIZE’资源中设定 cannotBackground 标志。

程序清单 2 - 12 显示的是一个为响应空事件而执行空闲处理的过程。如果应用程序跑在前台并且当前活动窗口是一个文稿窗口，该代码段则调用 TextEdit 过程 TEIdle。在编辑记录引用的文字内的插入点上，TEIdle 过程会产生一个闪烁的插入记号。(该应用程序使用 TextEdit 在其文稿窗口内显示文字；如果您不在您的文稿窗口使用 TextEdit，则应提供您自己的例程来闪烁插入记号。）如果当前活动窗口是一个不定模式对话框，DoIdle过程将调用对话管理器函数DialogSelect在对话框的某个可编辑文字项目中闪烁插入记号。

程序清单 2 — 12 处理空事件

```
PROCEDURE DoIdle (event: EventRecord);
VAR
    window:         WindowPtr;
    myData:         MyDocRecHnd;
    windowType:     Integer;
    itemHit:        Integer;
    result:         Boolean;
```

```
BEGIN
    window := FrontWindow;
    {determine the type of window--document, modeless, etc.}
    windowType := MyGetWindowType(window);
    CASE windowType OF
        kMyDocWindow:
            IF (NOT gInBackground) THEN
            BEGIN
                myData := MyDocRecHnd(GetWRefCon(window));
                TEIdle(myData^^.editRec);
            END;
        kMyGlobalChangesID:
            result := DialogSelect(event, window, itemHit);
    END; {of CASE}
END;
```

处理操作系统事件

操作系统事件包括挂起，恢复执行，和鼠标移动事件。您的应用程序由于其处理状态的改变，所以需要接收挂起和恢复执行事件。在给 WaitNextEvent 的 MouseRgn 参数中指定一个非空区域，每当光标处于该指定区域之外时，您的应用程序可以请求事件管理器返回鼠标移动事件。如果您指定了一个空区域或一个在 MouseRgn 参数中的 NIL 区域间接指针，事件管理器将不报告鼠标移动事件。

您的应用程序通过检验事件记录来判断它接收到的是哪个事件并获取与事件相关的附加信息。

操作系统事件记录中的 what 域包含 osEvt 常数。

操作系统事件记录中的message域包含的信息可以表明该事件是一个挂起事件还是一个恢复执行或鼠标移动事件。message 域还可以表明当应用程序继续执行时是否要求剪贴板转换。在 message 域中的位给出了如下信息:

位	内容
0	如果是一个挂起事件则为 0 如果是一个恢复执行事件则为 1
1	如果不要求剪贴板转换则为 0 如果要求剪贴板转换则为 1
2 – 23	保留
24 – 31	如果是一个挂起或恢复执行事件则是 suspendResumeMessage 如果是一个鼠标移动事件则是 mouseMovedMessage

注意，您需要检验message域的位24 - 31，以便判断您接收到的是何种操作系统事件。message域中的位24 - 31包含下列两常数之一：

```
CONST suspendResumeMessage = $01;    {suspend or resume event}
      mouseMovedMessage    = $FA;    {mouse-moved event}
```

如果事件是一个挂起或恢复执行事件，您需要检验位0以确定该事件是一个挂起事件还是一个恢复执行事件。只有当位24 - 31表明该事件是一个挂起或恢复执行事件时，位0才有意义。您可以使用resumeFlag常数来确定该事件是一个挂起还是一个恢复执行事件。如果该事件是一个恢复执行事件，您可以使用convertClipboardFlag常数来确定是否要求进行剪贴板转换，即从剪贴板向您的应用程序剪贴数据存储区转换。

```
CONST resumeFlag            = 1; {resume event}
      convertClipboardFlag = 2; {Clipboard conversion required}
```

每当用户执行一个拷贝或剪切操作时，您的应用程序将选定的数据拷贝到它的专用剪贴数据存储区；如果您的应用程序没有专用剪贴数据存储区，则应拷贝到剪贴板。如果您的应用程序使用的是专用剪贴数据存储区，则每当您的应用程序接收到一个挂起事件时，您都必需将您的专用剪贴数据存储区的数据转换到剪贴板。同样，当您的应用程序接收到一个恢复执行事件时，您需要从剪贴板（如果它已改变）中把所有数据转换。对于恢复执行事件，如果您的应用程序需要读取剪贴板的新内容，message域位 1 的值为 1。

程序清单2 - 13显示了一个响应操作系统事件的过程。

程序清单2 - 13 响应操作系统事件

```
PROCEDURE DoOSEvent (event: EventRecord);
BEGIN
    CASE BAnd(BRotL(event.message, 8), $FF) OF {get high byte}
        mouseMovedMessage:
            DoIdle(event);{mouse-moved same as idle for this app}
        suspendResumeMessage:
            DoSuspendResumeEvent(event);{handle supend/resume event}
    END;
END;
```

每当应用程序收到一个操作系统事件，DoEvent过程（显示于2 - 26页上的程序清单2 - 3中）就调用程序清单2 - 13中的DoOSEvent过程。DoOSEvent过程首先检验message域的高位字节以确定事件是一个移动鼠标、挂起，还是恢复执行事件，随后就调用一个应用程序定义的过程来处理事件。注意，大多数应用程序在处理事件时均要调整光标，要么调整光标响应鼠标移动事件，要么当收到任何类型的事件时均在它们的事件循环中调

整光标。本章中的代码段采用后一种方法，因而 DoOSEvent 过程简单地调用它的 DoIdle 过程来响应鼠标移动事件。下面两节将显示处理挂起，继续执行和鼠标移动事件的代码段。

响应暂停和恢复执行事件

当您的应用程序即将被切换到后台时，WaitNextEvent 函数返回一个挂起事件。当您的应用程序重新变成前台进程时，WaitNextEvent 返回一个恢复执行事件。

接收到一个挂起事件后，您的应用程序应该做好切换到后台的准备即停用前面的窗口，取消所有选定项的醒目显示，并隐藏所有浮动窗口。如果有必要，您的应用程序还应该将所有专用剪贴数据存储区转换成全局剪贴数据存储区。如果您的应用程序显示一个显示剪贴板内容的窗口，您也应隐藏该窗口，因为用户有可能在返回到您的应用程序之前改变剪贴板的内容。您的应用程序还可做其他任何为主切换作准备的必要的事情。然后，您的应用程序应调用 WaitNextEvent 来让出 CPU 以便让操作系统为其它进程安排执行时间。

收到一个恢复执行事件后，您的应用程序应该激活前端窗口并把所有窗口还原到上一次挂起事件时的状态。例如，您的应用程序应该显示滚动杆，还原原先起作用的所有选定项，并显示所有浮动窗口。如果有必要，您的应用程序还应拷贝剪贴板的内容并把数据转换到它的专用剪贴数据存储区。如果您的应用程序显示了一个显示剪贴板内容的窗口，您可以在读入剪贴板数据存储区后更新窗口内容。这样，您的应用程序就可以和用户对话了。

响应挂起或恢复执行事件通常要涉及到激活或失活窗口。如果您在您的应用程序的 'SIZE' 资源中设定了 acceptSuspendResumeEvents 标志和 doesActivateOnFGSwitch 标志，则您的应用程序在响应处理挂起和恢复执行事件时负责激活或失活您的应用程序窗口。

附 注:

如果您在您的应用程序的 'SIZE' 资源中设定了acceptSuspendResumeEvents标志而没有设定 doesActivateOnFGSwitch 标志，您的应用程序在收到一个挂起或恢复执行事件后会立即收到一个激活事件。在大多数情况下，您应该在您的应用程序的 'SIZE' 资源中同时设定 acceptSuspendResumeEvents 和 doesActivateOnFGSwitch 这两个标志。

您的应用程序可以使用剪贴数据存储区管理器函数InfoScrap,ZeroScrap,PutScrap 和 GetScrap 从剪贴板中读取数据或写入数据。请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书“剪贴数据存储区管理器”一章中的补充说明。

附　注：

如果您的应用程序不处理挂起和恢复执行事件(由其`SIZE'资源中的一个标志指示)，操作系统就不得不想法让您的应用程序执行剪贴板数据存储区强制功能以确保剪贴板的内容能从一个应用程序转送到另一个应用程序。这个过程增强了操作系统把前台应用程序切换到后台或把后台应用程序切换到前台所需的时间。

程序清单 2-14 显示了一个响应挂起和恢复执行事件的过程。DoSuspendResumeEvent 过程首先使用窗口管理器函数 FrontWindow 获取一个前端窗口指针。然后，它检验事件记录的 message 域的位 0 以确定该事件是一个挂起事件还是一个恢复执行事件。如果是一个恢复执行事件，程序将检查事件记录的 message 域的位 1 以确定它是否需要读剪贴板数据存储区的内容。如果需要读取，程序将调用一个应用程序定义的例程 MyConvertScrap，该例程读取剪贴数据存储区并把内容转换为它的专用剪贴数据存储区。它随后将专用全局标志 gInBackground 设定为 FALSE，以表明该应用程序不是在后台。然后再调用另一个应用程序定义的例程 DoActivate（显示于程序清单 2－10 中）来激活应用程序的前端窗口。

对于挂起事件，DoSuspendResumeEvent 过程调用应用程序定义的过程 MyConvertScrap，把它的专用剪贴数据存储区的内容拷贝到全局剪贴数据存储区。它随后把 gInBackground 这个专用全局标志设定为 TRUE，以表明应用程序处于后台。最后，它调用另一个应用程序定义的例程来停用应用程序的前端窗口。

程序清单 2－14　响应挂起和恢复执行事件

```
PROCEDURE DoSuspendResumeEvent (event: EventRecord);
VAR
    currentFrontWindow: WindowPtr;
BEGIN                                   {handle suspend/resume event}
    currentFrontWindow := FrontWindow;
    IF (BAnd(event.message, resumeFlag) <> 0) THEN
    BEGIN                           {it's a resume event}
        IF (BAnd(event.message, convertClipboardFlag) <> 0) THEN
          MyConvertScrap(kClipboardToPrivate);
        gInBackground := FALSE;
                                    {activate front window}
        DoActivate(currentFrontWindow, NOT gInBackground, event);
        MyShowClipboardWindow; {show Clipboard window if it was }
                                    { showing at last suspend event}
        MyShowFloatingWindows;      {show any floating windows}
    END
    ELSE
```

```
    BEGIN              {it's a suspend event}
        MyConvertScrap(kPrivateToClipboard);
        gInBackground := TRUE;
                       {deactivate front window}
        DoActivate(currentFrontWindow, NOT gInBackground, event);
        MyHideClipboardWindow; {hide Clipboard window if showing}
        MyHideFloatingWindows;  {hide any floating windows}
    END;
END;
```

您的应用程序跑在后台时可以接受处理时间并在台后执行任务，但您的应用程序不应与用户对话或执行可能使得前台进程响应降低的任务。

当您的应用程序正在后台执行任务时，如果您需要把某个特殊事件通知给用户，您就应该使用通知管理器安排一个通知请求。请参阅《Inside Macintosh: Processes》一书“通知管理器”一章中关于如何发送通知请求的实例。

响应鼠标移动事件

每当用户移动鼠标时、鼠标驱动程序和事件管理器以及您的应用程序就会负责向用户提供反馈信息。鼠标驱动程序执行低级功能，如不断地登记鼠标的位置和状态并在一个全局变量中维持鼠标的当前状态。

当用户移动鼠标时，用户希望将光标移到屏幕上一个对应的相关位置。鼠标驱动程序的低级中断例程可将移动鼠标映射到屏幕上的相关位置。每当用户移动鼠标时，鼠标驱动程序的低级中断例程将会移动屏幕上显示的光标，并将光标的校准点与新的鼠标位置对齐。校准点是指鼠标驱动程序用来使光标与鼠标位置对齐的点。

您的应用程序负责设定光标的初始形状，在WaitNextEvent返回后还原光标，同时还可以为更好地适应您的应用程序而对光标形状作适当的修改。例如，大多数应用程序通常在一个文稿的文字编辑区域中把光标设定为I型光标，而当光标在一个文稿的滚动杆中时将光标改为箭头光标。如果用户将光标移到了您在WaitNextEvent函数的mouseRgn参数中指定的区域之外，您的应用程序就可以通过请求事件管理器报告鼠标移动事件来达到上述这种效果。

鼠标驱动程序和您的应用程序控制着光标的形状和外观。光标可以是一个 256 位的图像，由一个 16 × 16 位方块定义。鼠标驱动程序显示当前光标，您的应用程序可以使用各种光标处理例程来更改光标 例如，SetCursor过程)。

图2-13显示了标准的箭头光标。您可以使用InitCursor过程将光标初始化为标准的箭头光标。在图2-13中，箭头光标的校准点是位置 1, 1)。请参阅《Inside Macintosh

绘图》(*Inside Macintosh: Imaging*) 一书中关于光标处理例程以及您的应用程序怎样定义它自己的光标的详细介绍。

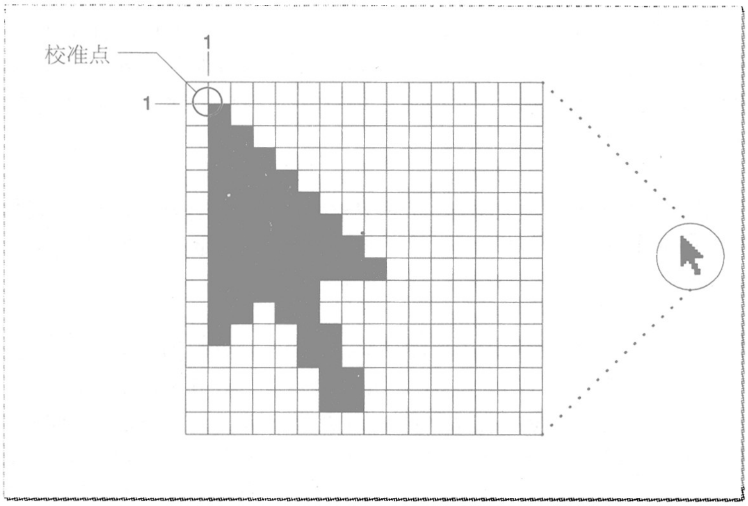


图 2-13　标准的箭头光标

图 2 - 14 显示了另外 4 种您的应用程序可用的常见光标：I 型光标，十字准线光标，加号光标和手表光标。



图 2-14　I 型光标、十字准线光标、加号光标和手表光标

I 型光标、十字准线光标、加号光标和手表光标被作为资源来定义，并且您的应用程序可以通过给GetCursor函数指定相应的资源标识符来获取这些光标的任一间接指针。如下常数为 I 型光标，十字准线光标，加号光标，和手表光标指定了相应的资源标识符：

```
CONST iBeamCursor = 1; {used in text deiting}
      crossCursor = 2; {often used for manipulating graphics}
      plusCursor  = 3; {often used for selecting fields in}
                       { an array}
      watchCursor = 4; {used to mean a lengthy operation }
                       { is in progress}
```

您可以使用SetCursor过程或其它光标处理例程来改变光标的外观。您还可以定义您自己的光标，将它们储存在资源中，需要时，就可以在您的应用程序中使用它们。

当用户将光标移动到文稿中的不同区域时，您的应用程序通常需要改变光标的形状。您的应用程序可以使用光标移动事件来完成这项工作。您的应用程序在响应恢复执行事件时，还需调整光标。大多数应用程序在响应事件时都要通过事件循环调整一次光标。

当光标在函数WaitNextEvent的参数指定的区域之外时，您可以请求事件管理器报告鼠标移动事件。如果您给WaitNextEvent函数指定了一个空区域或一个NIL间接指针，WaitNextEvent将不报告鼠标移动事件。

如果您在函数WaitNextEvent的mouseRgn参数中指定的是一个非空区域，则每当光标移出该区域时， WaitNextEvent就会返回一个鼠标移动事件。例如，图2－15显示了一个文稿窗口。应用程序可能会定义两个区域：一个是包围着窗口文字区域的区域（I型标区域），和一个定义滚动杆及文字区域以外的其它所有区域的区域（箭头区域）。在为WaitNextEvent指定了I型区域后，鼠标驱动程序将不断地显示I型光标直到用户将光标移出该区域为止。



图2-15　箭头光标区域和I型光标区域

当用户将光标移出I型光标区域时，WaitNextEvent报告一个鼠标移动事件。然后您的应用程序可以将I型光标改变为箭头光标，并将mouseRgn参数改为由滚动杆和所有I型光标区域以外的其它区域定义的区域。现在光标就保持箭头状态，直到用户将光标移到该区域以外，并且您的应用程序收到一个鼠标移动事件之后，光标状态才会改变。

图2－16表示一个应用程序在收到一个鼠标移动事件后怎样将I型光标改变为箭头光标。

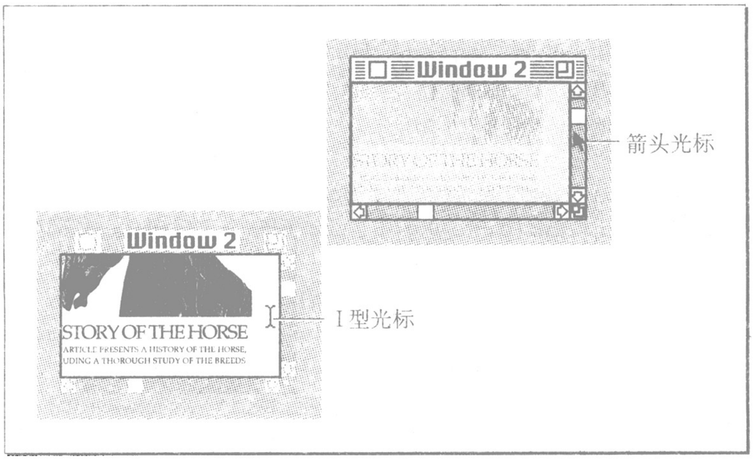


图 2-16　将光标从 I 型光标改变为箭头光标

注意，您的应用程序收到一个鼠标移动事件时应重新计算mouseRgn参数；否则，它将继续接收鼠标移动事件直至光标位置移出最初区域为止。

只要收到的不是高级事件，MyEventLoop 过程（显示在 2 - 24 页的程序清单 2 - 2 中）都将调用应用程序定义的过程 MyAdjustCursor 来调整光标。调整光标以后，如果事件是操作系统事件，DoEvent 过程将继续调用 DoOSEvent 过程。而 DoOSEvent 过程则调用 DoIdle 过程来处理鼠标移动事件。DoIdle 过程调用 TEIdle 仅仅是为了在文字编辑窗口中的闪烁插入记号。

程序清单 2-15　　显示了应用程序定义的例程 MyAdjustCursor。

程序清单 2-15　改变光标

```
PROCEDURE MyAdjustCursor (mouse: Point; VAR region: RgnHandle);
VAR
     window:                    WindowPtr;
     arrowRgn:                  RgnHandle;
     iBeamRgn:                  RgnHandle;
     iBeamRect:                 Rect;
     myData:                    MyDocRecHnd;
     windowType:                Integer;
BEGIN
window := FrontWindow;
{determine the type of window--document, modeless, etc.}
windowType := MyGetWindowType(window);
CASE windowType OF
     kMyDocWindow:
```

```
BEGIN
    {initialize regions for arrow and I-beam}
    arrowRgn := NewRgn;
    ibeamRgn := NewRgn;

    {set arrow region to large region at first}
    SetRectRgn(arrowRgn, -32768, -32768, 32766, 32766);

    {calculate I-beam region}
    {first get the document's TextEdit view rectangle}
    myData := MyDocRecHnd(GetWRefCon(window));
    iBeamRect := myData^^.editRec^^.viewRect;
    SetPort(window);
    WITH iBeamRect DO
    BEGIN
        LocalToGlobal(topLeft);
        LocalToGlobal(botRight);
    END;
    RectRgn(iBeamRgn, iBeamRect);
    WITH window^.portBits.bounds DO
        SetOrigin(-left, -top);
    {intersect I-beam region with window's visible region}
    SectRgn(iBeamRgn, window^.visRgn, iBeamRgn);
    SetOrigin(0,0);

    {calculate arrow region by subtracting I-beam region}
    DiffRgn(arrowRgn, iBeamRgn, arrowRgn);

    {change the cursor and region parameter as necessary}
    IF PtInRgn(mouse, iBeamRgn) THEN {cursor is in I-beam rgn}
    BEGIN
        SetCursor(GetCursor(iBeamCursor)^^);{set to I-beam}
        CopyRgn(iBeamRgn, region);{update the region param}
    END;

    {update cursor if in arrow region}
    IF PtInRgn(mouse, arrowRgn) THEN  {cursor is in arrow rgn}
    BEGIN
        SetCursor(arrow);          {set cursor to the arrow}
        CopyRgn(arrowRgn, region);{update the region param}
    END;
    DisposeRgn(iBeamRgn);
    DisposeRgn(arrowRgn);
END; {of kMyDocWindow}
```

```
            kMyGlobalChangesID:
                MyCalcCursorRgnForModelessDialogBox(window, region);
            kNil:
            BEGIN
                MySetRegionNoWindows(kNil, region);
                SetCursor(arrow);
            END;
        END; {of CASE}
    END;
```

MyAdjustCursor过程可以根据一个文稿窗口或不定模式对话框是否呈当前活动状态而正确地设定光标。

对于一个文稿窗口，程序清单2 - 15中的代码段定义了两个区域，由arrowRgn和iBcamRgn变量指定。如果光标在变量arrowRgn说明的区域内，该代码段则将光标设定为箭头光标并返回由arrowRgn说明的区域。同样地，如果光标处在变量iBeamRgn说明的区域内，该代码段则将光标设定为I型光标并返回由iBeamRgn说明的区域。

MyAdjustCursor过程是通过先将箭头区域设定为最大的可能区域的方法来计算这两个区域的。然后把文稿TextEdit显示矩形说明的区域设定为I型区域。该区域的面积一般相当于窗口的内容区域面积与滚动杆区域面积之差。(如果您的应用程序没有在其文稿窗口使用TextEdit，则将该区域设定为适宜于您的应用程序即可。) 然后程序调整I型标区域使得它仅包括窗口的可视区域中的内容区部分(例如，要把可能履盖窗口的任何浮动窗口考虑在内)。然后该代码段将I型标区域以外的整个屏幕设定为箭头区。

这样，过程就可以确定光标是处在那个区域并正确设定光标形状和区域参数。

对于不定模式对话框（例如全局改变不定模式对话框)，MyAdjustCursor过程调用一个应用程序定义的例程为不定模式对话框正确地调整光标。如果当前没有任何窗口被打开，MyAdjustCursor过程也相应地调整光标。

处理高级事件

高级事件提供了一个应用程序间通讯的方法。苹果事件是遵循苹果事件进程间通讯协议 (AEIMP) 的高级事件。在大多数情况下，如果您想与其它应用程序通讯，您应该使用苹果事件而不应定义您自己的高级事件。如果您计划使用苹果事件，请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中关于苹果事件的专门介绍，并参考本节关于事件管理器怎样报告高级事件的详细说明。

若要接收高级事件，您必须在您的应用程序的'SIZE'资源中设定适当的标志。如果您的应用程序将要接收高级事件，您必须设定isHighLevelEventAware标志。您的应用程序若要接收来自网络上的另一个电脑的高级事件，您必须设定localAndRemoteHLEvents标志。另外，若要接收来自另一个电脑的高级事件，必须共享您的应用程序，而且必须实现程序链接（Program Linking)。用户在Finder中选择您的应用程序以便共享您的应用程序，在文件菜单中选择共享并通过共享设置控制板使程序链接生效。

如果您在您的应用程序的'SIZE'资源中设定了isHighLevelEventAware标志，那么您的应用程序将按苹果事件格式接收Finder信息。您的应用程序可以使用Finder信息来确定哪个文件已打开或正在打印。如果您的应用程序发送或接收高级事件，您的应用必须响应所需的由Finder发送的苹果事件（打开应用程序，打开文稿、打印文稿，和退出应用程序）。

一个高级事件的事件记录的What域中包含kHighLevelEvent常数。

若要确定接收的高级事件的类型，您的应用程序需要检查事件记录的message和where域。对于高级事件来说，事件记录的这两个域有特殊的含义。

事件记录的 message 和 where 域共同定义所接收到的高级事件的具体类型。您的应用程序应把这些域解释为 OSType 数据类型，而非 LongInt 或 Point 数据类型。

message域包含高级事件的事件类。例如，版本管理器发送的苹果事件的事件类为'sect'。您可以定义您的应用程序自己专门的事件分组。如果您已在苹果电脑公司注册了您的应用程序名称，那么您也可以使用您的署名来定义属于您的应用程序的事件类。但要注意，苹果保留所有命名仅包含小写字母和非字母字符的事件类的使用权。

对于高级事件，事件记录的where域包含一个叫做事件标识符的第二消息标识符。事件标识符可以定义由事件类定义的事件类内的事件（或消息）的特殊类型。例如，由版本管理器发送的节读（Section Read）事件的事件类是'sect'，事件标识符是'read'。由Finder发送的打开文稿事件的事件类是'aevt'，事件标识符为'odoc'。您也可以定义您自己的事件类以及和您自己的事件类相对应的事件标识符集。例如，如果message域包含'biff'而where域包含'cmd1'，那么该高级事件表明的事件类型是由应用程序用署名'biff'定义的事件类中的'cmd1'定义的。

附 注:

如果您的应用程序支持苹果事件，您就可以调用AEProcessAppleEvent函数来确定收到的苹果事件的类型，而不必检查message和where域。

注意，因为高级事件记录中的 where 域被用来选定一个特殊种类事件（在 message 域决定的类内），所以高级事件记录不包含事件发生时鼠标的位置。您不应在解释 what 域之前解释 where 域，因为不同类的事件可能会包含重叠的事件标识符集。

高级事件和低级事件及操作系统事件不同，因为在调用 WaitNextEvent 时，有关高级事件的信息可能并不完全取决于返回到您的应用程序的事件记录。例如，您也许仍然需要知道究竟是另外的哪一个应用程序发送给您该高级事件，或者该应用程序还要发送给您什么数据。您的应用程序可以通过调用 AcceptHighLevelEvent 得到有关该高级事件的进一步信息。和高级事件有关的进一步信息包括:

- 确定事件的发送程序
- 唯一能够识别和该事件有关的请求的数字，或者将特定事件和上一事件的请求唯一地联系起来的数字
- 能够包含可选数据的数据缓冲区的地址和长度

如果要得到这个附加信息，在再次调用 WaitNextEvent 之前，您的应用程序必须调用 AcceptHighLevelEvent。按照惯例，需调用 AcceptHighLevelEvent 来表明您的应用程序希望处理该高级事件。

如果要接收苹果事件，应该调用 AEProcessAppleEvent 函数，而不是 AcceptHigh-LevelEvent 函数。苹果事件管理器根据应用程序的请求也析取一切和苹果事件有关的附加信息。本章讨论如何使用 AcceptHighLevelEvent 函数接收高级事件；请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中有关 AEProcessAppleEvent 函数的信息。

响应来自其它应用程序的事件

您可以根据事件记录中 what 域的值来识别高级事件。message 和 where 域进一步分清了高级事件的类型。只要适当，您的应用程序可以选择识别任意数目的事件。一些高级事件可能完全是由它们的事件记录来指定的，而另一些可能在一个可选缓冲区中包含了附加信息。如果想获得这些附加信息或者查找事件的发送程序，可以使用 AcceptHighLevel-Event 函数。

附 注

如果要响应一个苹果事件，可以按照《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中的说明使用苹果事件管理器。

程序清单 2-16 对如何响应一个高级事件作了说明。

程序清单 2-16 中的 DoHighLevelEvent 过程首先通过核查事件记录的 message 和 where 域确定所接收的高级事件的类型。然后，使用 AcceptHighLevelEvent 来获取和该事件有关的一切附加信息。这个特定的应用程序仅识别一类高级事件。如果该事件不属于这种类型，代码段就会假定它是一个苹果事件，并调用 AEProcessAppleEvent 来处理该事件。

一般来说，您事先无法知道可选数据缓冲区有多大，所以您可以先分配一个零长度缓冲区。如果调用 AcceptHighLevelEvent 后返回的是 bufferIsSmall 结果代码，您就可以重新调整缓冲区的大小。

程序清单 2-16 接收一个高级事件

```
PROCEDURE DoHighLevelEvent (event: EventRecord);
VAR
    myTarg:       TargetID;               {target ID record}
    myRefCon:     LongInt;
    myBuff:       Ptr;
    myLen:        LongInt;
    myErr:        OSErr;
BEGIN
    IF (event.message = LongInt(kMySpecialHLEventClass)) AND
       (LongInt(event.where) = LongInt(kMySpecialHLEventID)) THEN
    BEGIN
       {it's a high-level event that doesn't use AEIMP}
        myLen    := 0;          {start with a 0-byte buffer}
        myBuff  := NIL;
        myErr:=AcceptHighLevelEvent(myTarg,myRefCon, myBuff, myLen);
        IF myErr = bufferIsSmall THEN
        BEGIN
           myBuff := NewPtr(myLen); {allocate needed storage}
           myErr := AcceptHighLevelEvent(myTarg, myRefCon, myBuff,
                                          myLen);
           IF myErr = noErr THEN
              ; {perform any action requested by the event}
        END;
        IF myErr <> noErr THEN
          DoError(myErr);     {perform the necessary error handling}
    END
    ELSE
    BEGIN {otherwise, assume that the event is an Apple event}
        myErr := AEProcessAppleEvent(event);
        IF myErr <> noErr THEN
          DoError(myErr);{perform the necessary error handling}
    END;
END;
```

AcceptHighLevelEvent 函数返回和该事件有关的附加信息和数据。事件发送程序的标识符被作为第一参数返回，它是一个目标标识符记录。您可以通过审查该记录的域来确定发送事件的应用程序。目标 标识符记录中含有对话参考号，可以用来识别

与其他应用程序的连接通道，同时还可以识别发送程序的端口名称及位置名称。如果高级事件请求您返回信息，您可以使用目标标识符 记录中返回的信息，给请求的应用程序回送一个事件。请参阅 2-72 页的“确定高级事件的发送程序”和 2-73 页的“发送高级事件”中有关目标 标识符 的专门信息。

AcceptHighLevelEvent 的第二个参数，即参考常数参数，是唯一识别和事件有关的请求的数字，也是唯一识别特定事件是否和上一个事件有关的数字。如果您对该事件作出一个响应，您应该使用同样的参考常数值，这样事件的发送程序就可以将应答和原来的请求联系起来。

第三个参数指向的是与事件有关的附加数据。辅助缓冲区中的数据是由特定的高级事件定义的。AcceptHighLevelEvent 的第四个参数是输入的长度参数。它包含缓冲区的大小。如果不出现什么错误，输出的长度参数包含所接收的消息的大小。如果 AcceptHighLevelEvent 函数返回的是结果代码 bufferIsSmall，表明含尚未收到的消息的大小。

搜索指定的高级事件

有时，您不希望接收等待应用程序处理的下一个可用高级事件。您也许需要从应用程序的高级事件队列中选择一个高级事件。例如，您可能在处理其它高级事件之前，希望查找您以前发送的一个高级事件的回执。

您可以通过调用 GetSpecificHighLevelEvent 函数来选择一个特殊的高级事件。首先通过事件过滤函数考察应用程序高级事件队列中的一个事件，并确定它是不是您希望接收的那类事件。如果是，过滤函数便返回 TRUE。这表明您的过滤函数不希望审查更多的事件。如果过滤函数找到了一个符合类型要求的事件，它应该调用 AcceptHighLevelEvent 来检索该事件。当您的函数返回 TRUE 时，GetSpecificHighLevelEvent 函数本身也返回 TRUE。

如果您的过滤函数对高级事件队列中的一个事件返回的是FALSE，那么GetSpecific-HighLevelEvent 就会察看高级事件队列中的下一个事件，并执行您的过滤函数。如果过滤函数对队列中所有的高级事件都返回一个 FALSE，那么 GetSpecificHighLevel-Event 本身也向您的应用程序返回 FALSE。

下面为您提供了说明过滤函数的方法，您需要将该过滤函数的地址传送给 GetSpecificHigh-LevelEvent 函数：

```
FUNCTION MyFilter  (yourDataPtr : Ptr ;
                    msgBuff : HighLevelEventMsgPtr ;
                    sender : TargetID) : Boolean ;
```

当您的应用程序调用 GetSpecificHighLevelEvent 时，您向它传递一个参数，该参数给出了过滤函数搜索特殊事件的条件。GetSpecificHighLevelEvent 函数会将此信息传送到过滤函数的 yourDataPtr 参数中。另外，GetSpecificHighLevelEvent 函数还为过滤函数的 msgBuff 参数提供有关高级事件记录的信息以及 sender 参数提供有关发送程序的信息。

msgBuff 参数包含一个高级事件消息记录指针，它的结构如下:

```
TYPE      HighLevelEventMsg =
          RECORD
               HighLevelEventMsgHeaderLength:  Integer;
               version:                        Integer;
               reserved1:                      LongInt;
               theMsgEvent:                    EventRecord;
               userRefCon:                     LongInt;
               postingOptions:                 LongInt;
               msgLength:                      LongInt;
          END;

          HighLevelEventMsgPtr = ^HighLevelEventMsg;
```

当您调用GetSpecificHighLevelEvent，而它在执行您的过滤函数处理高级事件队列中的高级事件时，该高级事件消息记录的域由事件管理器来填写。然后，您可以将该信息和 youDataPtr 参数中的信息比较，确定该事件是否满足了您的需要。例如，youDataPtr 参数也许包含一个回执署名。您可以根据该事件记录的事件类测试它的值，信息包含在高级事件消息记录的 theMsgEvent 域中。

确定高级事件的发送程序

当您收到一个高级事件时，AcceptHighLevelEvent 返回的部分信息是事件发送程序的标识码。您可以使用该信息有选择地对其它应用程序的请求做出响应，或者弄清楚需要向哪个应用程序发送应答。有关发送程序的信息以目标 标识符 记录的形式提供，定义如下:

```
TYPE TargetID =
     RECORD
          sessionID:LongInt;          {session reference number}
          name:     PPCPortRec;       {sender's port name}
          location: LocationNameRec; {sender's location name}
          recvrName:PPCPortRec   ;    {reserved}
     END;
```

sessionID 域对应的是 PPC 工具箱创建的对话参考号。这是一个 32 位的数，是识别 PPC 工具箱和另一个应用程序对话 或联系）的唯一编码。name 和 location 域包含了发送程序的端口号码和位置名称。如果发送的应用程序和接收应用程序在同一个电脑上，您就可以通过调用 GetProcessSerialNumberFromPortName 来决定发送应用程序的进程序号。

发送高级事件

您可以使用 PostHighLevelEvent 函数来向另一个应用程序发送高级事件。如果要执行这一任务，您需要提供 6 条信息：

- 一个带有事件类和事件标识符的事件记录，其事件类和事件标识符需要正确指派
- 事件接收方的识别码
- 用来识别该特定事件通讯信息的唯一数字
- 可以包含可选数据的数据缓冲区
- 数据缓冲区的长度
- 决定事件传递方式的选项

附 注：
如果要发送一个苹果事件，请使用苹果事件管理器函数 AESend。苹果事件管理器使用事件管理器来传递苹果事件。请参阅《Inside Macintosh 应用程序间通信》（*Inside Macintosh: Interapplication Communication*）》一书中有关传递苹果事件的信息。

当您向同一电脑上的一个应用程序传递一个高级事件时，您可以采用下列 4 种方法之一指定其接收方：

- 使用端口名称和位置名称 在目标 标识符 记录中指定）来指定
- 使用对话参考号来指定
- 使用应用程序的创建者署名来指定
- 使用进程序号来指定

为了指定通过网络发送的高级事件，您只能使用接收应用程序的端口名称和位置名称，或它的对话参考号等两种方法。如果向同一电脑上的应用程序发送高级事件，您可采用上述 4 中方法中的任一种。

当您调用 PostHighLevelEvent 函数时，您应在 receiverID 参数中指定高级事件的接收方。如果要指定端口名称和位置名称，请在 receiverID 参数中提供一个目标 标识符 记录的地址。如果要指定一个进程序号，请在 receiverID 参数中提供它的地址。如果要指定一个对话参考号或署名，请提供 receiverID 参数中的数据。

当您应答一个高级事件时，因为可以使用从 AcceptHighLevelEvent 收到的目标标识符记录和目标标识符记录中包含的对话参考号，或进程序号 如果接收进程是本地进程），

所以很容易识别接收程序。注意，使用对话参考号的应答来响应高级事件是最快的方法。

如果您不是在应答上一个事件，您需要自己决定目标应用程序的识别码。您可以使用这几种方法中的一种来完成这项工作。如果目标应用程序跑在同一个电脑上，您可以通过调用GetProcessInformation 函数来搜索该应用程序的创建者署名，或它的进程序号。请参阅《Inside Macintosh进程》(*Inside Macintosh: Process*) 一书的“进程管理器”一章中有关GetProcessInformation 函数的详细说明，以及如何在本地电脑上使用它生成所有打开进程的进程序号的实例。

如果需要向其发送高级事件的应用程序在一个远程机上，您需要通过它的对话参考号码，或端口名称和位置名称来识别它。您可以通过调用PPCBrowser函数来让用户浏览一个特殊端口。您还可以调用 IPCListPort 函数来得到用目标PPC工具箱登记的所有端口列表。请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书的“程序间通讯工具箱”一章中对这两个函数的说明。

正如上面所说，您可以用4种方法中的任一种来识别高级事件的发送程序。程序清单2-17 说明了如何使用本地机上应用程序的创建者署名来向它发送高级事件。在这个例子中，一个应用程序将一个高级事件发送到另一个应用程序，接收程序的创建者署名是'boff'。具体要发送的高级事件用事件类 'boff' 和事件标识符 'cmd1' 来标识。

程序清单2-17　　高级事件的发送程序

```
PROCEDURE MyPostTest;
VAR
    myEvent:      EventRecord;      {an event record}
    myRecvID: OSType;               {receiver ID}
    myOpts:       LongInt;          {posting options}
    myErr:        OSErr;
BEGIN
    myEvent.what   := kHighLevelEvent;
    myEvent.message  := LongInt ('boff');          {event class}
    myEvent.where  := Point (LongInt ('cmd1'));    {event ID}
    {the receiver is identified by its signature and }
    { a return receipt is requested}
    myOpts  := receiverIDisSignature + nReturnReceipt;
    myRecvID  := 'boff';              {receiver's signature}
    myErr := PostHighLevelEvent (myEvent, Ptr (myRecvID), 0, NIL, 0,
                        myOpts);
    IF myErr <> noErr THEN
        DoError (myErr);
END;
```

这个例子使用 PostHighLevelEvent 函数，没有发送附加数据，所以发送应用程序将 NIL 作为数据缓冲区的指针，并设定缓冲区长度为 0。myOpts 变量指定了传递选项。

传递选项有两类：传递选项以及和 receiverID 参数有关的选项。您可以指定一个或多个输送选项，来表明您是否希望其它应用程序下次接收该事件，以及您是否希望其它应用程序收到该事件后进行确认。您使用和 receiverID 有关的参数来表明您如何指定该事件的接收程序。用以下常数为各种传递选项：

```
常数 nAttnMsg              = $00000001;{give this message priority}
    nReturnReceipt         = $00000200;{return receipt requested}
    receiverIDisTargetID   = $00005000;{ID is port name and location name}
    receiverIDisSessionID  = $00006000;{ID is PPC session ref number}
    receiverIDisSignature  = $00007000;{ID is creator signature}
    receiverIDisPSN        = $00008000;{ID is process serial number}
```

当您在 receiverID 参数中指定了接收应用程序时，您可以使用这些常数按端口名称和位置名称、对话参考号、进程序号或署名来指定事件的接收程序。所有这些规范均允许您向本地机上的另一个应用程序发送事件。例如，在程序清单 2-17 中，myOpts 参数表明接收程序是用它的创建者署名来识别的，而 myRecvID 变量包含接收方的创建者署名。如果要给网络上电脑的应用程序发送事件，您只能使用对话参考号或端口名称及位置名称来指定接收方。

当您使用端口名称和位置名称指定接收程序时，请在传递选项参数中使用 receiverIDisTargetID 常数，并在 receiverID 参数中指定一个目标标识符地址。

```
TYPE    TargetID =
        RECORD
            sessionID:   LongInt;         {unused for posting}
            name:        PPCPortRec;      {recipient's port name}
            location:    LocationNameRec;{recipient's port loc}
            recvrName:   PPCPortRec;      {unused for posting}
        END;
```

当您传送一个目标标识符时，只需要指定 name 和 location 域。您可以使用 IPCListPort 函数列出全部现有端口名称和一些相关信息。这些信息会告诉您端口是否接受位置名称指定的电脑上的通讯服务。请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书的“程序间通讯工具箱”一章中有关如何使用 IPCListPort 函数的信息。

您也可以使用 PPCBrowser 函数来填写一个标识符记录。程序清单 2-18 说明了如何使用 PPCBrowser 函数来发送高级事件。在这个例子中，发送者希望确定一个字典的位置，并请求该字典应用程序向它回送一个字的定义。

程序清单 2-18 使用 PPCBrowser 函数发送一个高级事件

```
FUNCTION MyPostWithPPCBrowser (aTextPtr: Ptr; textlength: LongInt): OSErr;
VAR
     myHLEvent:          EventRecord;
     myErr:              OSErr;
     myNumTries:         Integer;
     myPortInfo:         PortInfoRec;
     myTarget:           TargetID;
BEGIN
     {use PPCBrowser to get the target}
     myErr := PPCBrowser('Select an Application', 'Application', FALSE,
                         myTarget.location, myPortInfo, NIL, '');
     IF myErr = NoErr THEN
     BEGIN
          {copy port name into myTarget.name}
          myTarget.name := myPortInfo.name;

          myHLEvent.what := kHighLevelEvent;
          myHLEvent.message := LongInt('Dict');
          myHLEvent.where := Point(LongInt('Defn'));

          {if a connection is broken, then sessClosedErr is returned to }
          { PostHighLevelEvent; to reestablish the connection, just post }
          { the event one more time}
          myNumTries := 0;
          REPEAT
               myErr:= PostHighLevelEvent(myHLEvent,@myTarget, 0, aTextPtr,
                              textlength, receiverIDisTargetID);
               myNumTries := myNumTries + 1;
          UNTIL (myErr <> sessClosedErr) OR (myNumTries > 1);
     END;
     MyPostWithPPCBrowser := myErr;      {return any error}
END;
```

在程序清单 2-18 中，应用程序定义的函数使用 PPCBrowser 函数来显示一个对话框，请求用户选定一个字典。(请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中有关 PPCBrowser 函数更全面的信息。) 如果用户选定了一个字典，该代码段就会向该字典应用程序发送一个高级事件，查询选定文字的定义。注意，发送和接收的应用程序的定义查询都属于 'Dict' 类的事件，事件标识符是 'Defn'。只有当不存在合适的苹果事件时，才必须定义专用协议。

附 注:
您应该避免为了试图共享一块数据,而向接收应用程序传送间接指针。较好的办法是将有关数据放入一个缓冲区（如程序清单 2-18 所示）并传送缓冲区的地址。如果您必须通过传送间接指针来共享数据的话，应确保该数据块是在系统堆中。

如果成功传递了高级事件，PostHighLevelEvent 就会返回结果代码 noErr，它仅表明事件成功地传送到了 PPC 工具箱。您的应用程序需要调用另一个事件管理器的例程（EventAvail，GetNextEvent 或 WaitNextEvent）以便另一个应用程序能够有机会收到该事件。

您所发送的事件可能会要求其它应用程序通过给您的应用程序返回一个高级事件,来返回某些信息。您可以使用 GetSpecificHighLevelEvent 对响应进行扫描。如果您的应用程序必须等待这个事件,比如希望显示一个手表光标,或者采取对您的应用程序适当的动作等。如果您的应用程序一直没有收到对事件的响应，您也可能希望执行一个超时机制。

请求回执

当您发送一个高级事件时,您可以通过将 nReturnReceipt 常数作为一个传递选项,要求返回一个回执。即要求事件管理器向您的应用程序发送一个高级事件,告诉您其它应用程序是否收到了您发出的事件。注意,这不一定意味着其它应用程序执行您可能要求它的动作。

回执是一个高级事件，它的事件类和事件标识符 用这两个常数表示:

```
CONST  HighLevelEventMsgClass   = 'jaym';
        rtrnReceiptMsgID         = 'rtrn';
```

回执由事件管理器投寄到接收应用程序所在的电脑(不是由接收应用程序本身)。数据缓冲区和回执无关。但是，执行投寄任务的事件管理器将会把高级事件记录的 modifiers 域设定为下列值之一:

```
CONST  msgWasNotAccepted         = 0;
        msgWasFullyAccepted       = 1;
        msgWasPartiallyAccepted = 2;
```

msgWasNotAccepted 常数表明，您的事件没有被接收应用程序接收。这意味着接收应用程序已经注意到您的事件到来（通过 WaitNextEvent），但是并没有调用 AcceptHighLevelEvent 来接收该事件。msgWasFullyAccepted 常数表明接收应用程序确实调用了 AcceptHighLevelEvent，并在可选数据缓冲区中检索了所有的数据。msgWas-PartiallyAccepted 常数表明接收应用程序调用了 AcceptHighLevelEvent但是应用程序的数据缓冲区太小,无法容纳应用程序所发送的数据,接收应用程序在检索缓冲区的其余部分前，调用了 WaitNextEvent。

注意，回执并不显示接收应用程序的标识码。您需要调用 AcceptHighLevelEvent 来确定事件管理器是代表谁向您发送一个特定回执的。当 AcceptHighLevelEvent 成功返回时，sender 参数包含了一个目标标识符 记录，其域是为接收应用程序填写的。有了回执，msgLen 参数则为 0，msgBuff 参数是 NIL，megRefCon 参数包含了原始高级事件发送程序（即您的应用程序）独有的 refCon 参数。

处理苹果事件

如果您的应用程序使用高级事件，它必须对 Finder 发送的必要苹果事件响应。4 种必要苹果事件包括打开应用程序、打开文稿、打印文稿及退出应用程序。请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中有关如何处理必要苹果事件的信息。

当您的应用程序收到一个高级事件（如事件记录的 what 域中的 kHighLevelEvent 常数所示)，而且您的应用程序支持苹果事件时，请调用 AEProcessAppleEvent 函数 AEProcessAppleEvent 函数为您的应用程序提供了便利的方法，来识别苹果事件的事件类及事件标识符，并引导苹果事件管理器调用应用程序中处理苹果事件的代码。

如果要向其它应用程序发送苹果事件，请调用 AESend 函数。

为了保证与其它 Macintosh 应用程序的兼容性及顺畅的交互作用，您应该尽可能使用苹果事件协议来处理高级事件。如果能向其它应用程序发送苹果事件并从其它应用程序接收苹果事件，那么其它应用程序便可以和您的应用程序相互通讯，从而为用户提供更强的性能。

请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中有关如何发送和接收苹果事件的全面信息。

事件管理器参考资料

本节介绍了事件管理器和操作系统事件管理器的数据结构和例程。另外还介绍了 'SIZE' 资源。

数据结构

本节讲述事件记录、目标标识符、高级事件消息记录和操作系统事件队列的结构。事件管理器使用事件记录返回有关事件的信息。您可以使用一个目标标识符记录来指定或识别与您的应用程序通讯的另一个应用程序或进程的地址。如果您的应用程序提供了一个过滤函数作为GetSpecificHighLevelEvent函数的参数，那么您的过滤函数将在一个高级事件消息记录中接收到有关高级事件的信息。

事件记录

当您的应用程序使用一个事件管理器的例程来检索事件时，事件管理器通过一个事件记录返回有关被检索事件的信息。

```
TYPE      EventRecord =
          RECORD
              what:          Integer;      {event code}
              message:       LongInt;      {event message}
              when:          LongInt;      {ticks since startup}
              where:         Point;        {mouse location}
              modifiers:     Integer;      {modifier flags}
          END;
```

事件记录域说明

what

what 域表示接收到的事件类型。事件的类型可以用下列常数来识别

```
CONST
nullEvent        = 0; {no other pending events}
mouseDown        = 1; {mouse button pressed}
mouseUp          = 2; {mouse button released}
keyDown          = 3; {key pressed}
keyUp            = 4; {key released}
autoKey          = 5; {key repeatedly held down}
updateEvt        = 6; {window needs updating}
diskEvt          = 7; {disk inserted}
activateEvt      = 8; {activate/deactivate window}
osEvt            = 15;{operating-system event-- }
                      { resume, suspend, or }
                      { mouse-moved}
kHighLevelEvent= 23;{high-level event}
```

注意，在System 7 中，值为 9～14 的事件类型没有定义，保留它们以备苹果公司将来使用。同时，what 域的所有其它值也被保留下来，以供苹果公司使用。

message 与事件有关的附加信息。对这一信息的解释取决于事件类型。现将每个事件类型的 message 域内容总结如下：

事件类型	事件消息
空事件、鼠标释放、鼠标按下	未定义
键释放、键按下、自动键	低位字中的字符代码和模拟键代码。对于苹果桌面总线自动键 ADB）键盘，高位字的低位字节含有键盘事件发生时的 ADB 键盘地址。高位字中的高位字节保留备用。
更新、激活	指向更新、激活或失活窗口的指针。
磁盘插入	驱动器号在低位字，文件管理器结果代码在高位字。
恢复执行	suspendResumeMessage 常数在位 24-31 之间，位 0 中的 1 表示事件是一个恢复执行事件。位 1 中的 1 或 0 表示剪贴板转换是否需要，位 2-23 保留。
挂起	suspendResumeMessage 常数在位 24-31 之间，位 0 中的 0 表示事件是一个挂起事件。位 1 未定义，位 2-23 保留。
鼠标移动	mouseMoveMessage 常数在位 24-31 之间，位 2-23 保留，位 0 和位 1 未定义。
高级事件	高级事件所属的事件类。高级事件的 message 和 where 域定义了接收高级事件的特殊类型。

when when 域指示了事件发送的时间 从系统启动开始计时)。

where 对于低级事件和操作系统事件，where 域包含光标在事件投寄时的位置 以全局坐标表示)。对于高级事件，where 域包含一个第二事件标识符：事件标识符。事件标识符 在高级事件的 message 域定义的事件类中定义了事件的特定类型。对于高级事件，您应该将 where 域解释为具有 OSType 数据类型，而不是 Point 数据类型。

modifiers　　modifiers 域包含修正键和鼠标按钮在事件发送时的状态信息。对激活件，该域还指示窗口是否应该激活或失活。在 System 7 中，它还指示一个鼠标按下事件是否将某个应用程序切换到了前台。

每个修正键在事件记录的 modifiers 域中都用一个特定的位来代表。2-20 页　　的图 2-5 介绍了解释 modifiers 域的方法。修正键包括 Option、Command、　CapsLock、Ctrl 和 Shift 键。如果您的应用程序为这些键中任一个键（和其　　它键的组合）赋予了特殊的意义，或当按下鼠标按钮时，您可以测试 modifiers 域的状态以确定应用程序应该执行的动作。例如，您可以使用这个信息来确定用户是否同时按了 Command 键和另一个键来进行菜单命令选择

目标标识符记录

当您向另一个应用程序发送一个高级事件时，您可以使用目标标识符 记录来指定事件的接收程序。当您收到一个高级事件时，AcceptHighLevelEvent 函数使用一个目标标识符记录返回有关事件发送程序的信息。

TargetID 数据类型定义了目标标识符记录。

```
TYPE TargetID =
     RECORD
          sessionID:    LongInt;            {session reference number}
          name:         PPCPortRec;         {port name}
          location:     LocationNameRec;    {location name}
          recvrName:    PPCPortRec;         {reserved}
     END;
```

域说明

sessionID　　对于您的应用程序接收到的高级事件，该域包含 PPC 工具箱创建的对话参考号。这是一个 32 位的数字，是识别 PPC 工具箱和其它应用程序对话（或联系）的唯一编码。您的应用程序在向另一个进程发送高级事件时不使用该事件。(如果需要发送一个高级事件（该事件可以用对话参考号指定接收程序），请在 receiverID　参数中提供一个对话参考号指针，并在 PostingOptions 参数中的 PostHighLevelEvent 选项上使用 receiverIDisSessionID 常数。)

name　　对于您的应用程序收到的高级事件，该域包含了一个端口记录，它指定了发送高级事件的进程端口名称。在向本地机或联网电脑的进程发送高级事件时，您可以在该域提供一个PPC 端口记录，并在该记录中指定接收进程的端口名称。

	如果发送应用程序和接收应用程序在同一个电脑上，您可以调用 GetProcessSerialNumberFromPortName 函数来确定发送应用程序的进程序号。
location	对于您的应用程序收到的高级事件，该域包含一个位置名称记录，该记录可以用来识别发送高级事件进程的位置名称。在向本地机或联网电脑的进程发送高级事件时，您可以在该域中提供一个位置名称记录，并在该记录中指定位置名称。
recvrName	保留域。

高级事件消息记录

您可以调用 GetSpecificHighLevelEvent 函数和事件过滤函数，在应用程序的高级事件队列中搜索一个特定的高级事件。您的过滤函数收到一个高级事件消息记录的指针，它包含有关高级事件的记录。(请参阅 2-114 页的“搜索高级事件队列的过滤函数”中有关如何定义过滤函数的信息。)

HighLevelEventMsg 数据类型定义了高级事件消息记录的结构。

```
TYPE HighLevelEventMsg =
    RECORD
        HighLevelEventMsgHeaderLength:      Integer;
        version:                            Integer;
        reserved1:                          LongInt;
        theMsgEvent:                        EventRecord;
        userRefCon:                         LongInt;
        postingOptions:                     LongInt;
        msgLength:                          LongInt;
    END;
```

域说明

HighLevelEventMsgHeaderLength
为事件管理器保留

version	为事件管理器保留
reserved1	为事件管理器保留
theMsgEvent	高级事件的事件记录。事件过滤函数可以比较这个事件记录的域来确定该高级事件是不是所需要的事件。如果过滤函数找到了需要的事件，它应该调用 AcceptHighLevelEvent 来接收该事件，并从高级事件队列中删除该事件，然后将 TRUE 作为它的函数结果返回。

userRefCon	标识和事件有关的通讯信息的唯一数字
postingOptions	为事件管理器保留
msgLength	为事件管理器保留

事件队列

事件队列是操作系统事件管理器拥有的一个标准的 Macintosh 操作系统队列。在操作系统事件队列中，只储存了鼠标释放、鼠标按下、键释放、键按下和自动键事件。在大多数情况下，您的应用程序不应该直接存取事件队列。通常您应该调用 WaitNextEvent 函数，它可以从这个队列及其它来源检索事件。

事件队列由队列头指针及其后面真正的队列输入组成。事件队列和所有标准的 Macintosh 操作系统事件队列一样，具有相同的头指针。Qhdr 数据类型定义了队列首部。

```
TYPE    QHdr =
        RECORD
            qFlags:  Integer;          {queue flags}
            qHead:   QElemPtr;         {first queue entry}
            qTail:   QElemPtr;         {last queue entry}
        END;
```

EvQEl 数据类型定义了操作系统事件队列的一个入口。

```
TYPE EvQEl =
    RECORD
        qLink:         QElemPtr;     {next queue entry}
        qType:         Integer;      {queue type (ORD(evType))}
        evtQWhat:      Integer;      {event code}
        evtQMessage:   LongInt;      {event message}
        evtQWhen:      LongInt;      {ticks since startup}
        evtQWhere:     Point;        {mouse location}
        evtQModifiers:Integer;       {modifier flags}
    END;
```

事件队列中的每个入口以 4 个字节的标志开始，后面是下一队列入口的一个指针。标志由操作系统事件管理器维护，并在该管理器之内。队列入口由指针链接，EvQEl 数据类型的第一域（代表一个队列入口的结构）以下一个队列入口的指针开始。这样，您就不能直接存取使用 EvQEl 数据类型的标志。

事件管理器例程

事件管理器包括接收事件、接收和发送高级事件，以及搜索特殊高级事件的例程。事件管理器还提供了进行进程序号和端口名称转换、获取鼠标按钮状态的信息、读键盘，以及取得定时信息的例程。

接收事件

您可以调用 WaitNextEvent 或 GetNextEvent 函数在事件管理器中检索一个事件，并从事件流中取消该事件。但是，为了对多任务提供较强的支持，只要可能，您都应该使用 WaitNextEvent 函数，而不是 GetNextEvent。您还可以通过调用 EventAvail 函数浏览一个事件，而不用从事件流中取消它。另外，您可以调用 AcceptHighLevel-Event 函数来得到和一个高级事件有关的附加信息，调用 GetSpecificHighLevelEvent 来搜索一个特殊的高级事件。

FlushEvents 过程可以取消操作系统事件队列中的所有低级事件。但一般来说，您的应用程序不应该使事件队列变空。

您可以通过 SystemClick 过程在必要时将事件发送给桌面附件。事件管理器可以调用 SystemTask 和 SystemEvent 例程，您的应用程序通常不需要调用这两个例程。

通常，您可以使用工具箱事件管理器检索事件流中的事件。如果您只对储存在操作系统事件队列中的事件感兴趣，可通过设定事件类型屏蔽参数屏蔽除键盘、鼠标和磁盘插入事件以外的所有事件，然后调用使用工具箱事件管理器检索这些事件。您也可以选择使用操作系统事件管理器的例程来执行这一任务。

操作系统事件管理器提供了两个函数：GetOSEvent 和 OSEventAvail。您可以调用这两个函数检索操作系统事件队列中的事件。但在大多数情况下，您的应用程序不需要调用这两个函数。

如果您的应用程序需要接收键释放事件，您可以通过 SetEventMask 过程改变应用程序的系统事件类型屏蔽参数。GetEvQHdr 函数返回操作系统事件队列头指针。

WaitNextEvent

您可以通过 WaitNextEvent 函数从事件管理器中每次检索一个事件。

```
FUNCTION WaitNextEvent (eventMask: Integer;
                      VAR theEvent: EventRecord; sleep: LongInt;
                      mouseRgn: RgnHandle): Boolean;
```

eventMask　　一个指明返回事件类型的值；该参数可以解释为事件类型屏蔽参数常数的和。您可以使用这些常数来指定事件类型屏蔽参数。

```
CONST
mDownMask      = 2;      {mouse-down event  (bit 1)}
mUpMask        = 4;      {mouse-up event    (bit 2)}
keyDownMask    = 8;      {key-down event    (bit 3)}
keyUpMask      = 16;     {key-up event      (bit 4)}
autoKeyMask    = 32;     {auto-key event    (bit 5)}
updateMask     = 64;     {update event      (bit 6)}
diskMask       = 128;    {disk-inserted event(bit 7)}
activMask      = 256;    {activate event    (bit 8)}
highLevelEventMask
               = 1024;   {high-level event  (bit 10)}
osMask         = -32768;{operating-system  (bit 15)}
```

如果需要接收所有的事件，您可以将 everyEvent 常数指定为事件类型屏蔽参数。

```
CONST
everyEvent  = -1;      {every event}
```

如果队列中没有符合指定了类型的事件，WaitNextEvent 便返回一个空事件。WaitNextEvent 根据 eventMask 参数和事件优先级确定下一个可用的事件。

被事件类型屏蔽参数屏蔽掉的事件将仍然保留在事件流中，直至被某个应用程序检索出去为止。操作系统事件队列中的低级事件一直保留在队列中，直至它们被您的应用程序或其它应用程序检索到，或者队列满员为止。当队列排满时，操作系统事件管理器就开始从队列中删除最旧的事件。

theEvent　　一个指定类型或多个指定类型的下一个可用事件。WaitNextEvent 函数将返回的事件从事件流中删除，并在一个事件记录中返回有关该事件的信息。事件记录包括若要接收高级事件，您必须在您的应用程序的 'SIZE' 资源中设定适当的标志。如果您的应用程序将要接收高级事

一个指定类型或多个指定类型的下一个可用事件。WaitNextEvent 函数将返回的事件从事件流中删除，并在一个事件记录中返回有关该事件的信息。事件记录包括接收到的事件类型和其它信息。请参阅 2-79 页的“事件记录”中有关事件记录域的说明。

除了事件记录外，高级事件也可以包含附加的数据；您可以调用 AcceptHighLevelEvent 或 AEProcessAppleEvent 函数来获取与这些事件有关的附加数据。

sleep 指表明您的应用程序在没有等待处理的事件（除了空事件外）时，愿意释放处理器的时间，用 Tick （一个 Tick 大约等于 1/60 秒）数表示。如果您为 sleep 参数指定一个大于 0 的值，您的应用程序就会在指定的时间内，或在一个事件发生前释放处理器。

通常，您应该为 sleep 参数指定一个大于 0 的值，以便后台进程能够接受处理时间。如果您的应用程序具有文字编辑能力，您就不应该将 sleep 的参数值设定成大于GetCaretTime 返回的Tick 数。当达到指定时间后，如果仍然没有需要您的应用程序来处理的事件WaitNextEvent 便给参数 theEvent 返回一个空事件。

mouseRgn 一个区域间接指针。在它指定的区域中，移动鼠标不会引起鼠标移动事件。换句话说，只有当光标出了指定区域，您的应用程序才会收到一个鼠标移动事件。您应该用全局坐标指定该区域。如果您传送了一个空区域或 NIL 区域间接指针，事件管理器便拒绝向您的应用程序报告鼠标移动事件。注意，当您的应用程序收到一个鼠标移动事件后，它应该重新计算 mouseRgn 参数，否则，只要光标位置在原来的 mouseRgn 之外，它将继续收到鼠标移动事件。

说明

如果 WaitNextEvent 函数返回的事件是一个空事件或者它截取了该事件，WaitNext-Event 将返回 FALSE 作为它的函数结果；否则，WaitNextEvent 将返回 TRUE。WaitNextEvent 函数调用操作系统事件管理器的SystemEvent 函数来确定事件应该由应用程序处理还是由操作系统处理。

如果没有事件等待您的应用程序处理，WaitNextEvent将在一个指定长度的时间内等待事件。（在这个时间内，CPU时间可以分配给后台进程）。如果发生了一个事件，它作为 theEvent 参数的值被返回，WaitNextEvent 同时返回函数结果 TRUE。如果指定的时间到后，仍没有等待应用程序处理的事件，WaitNextEvent 将给 theEvent 返回一个空事件以及函数结果 FALSE。

在给您的应用程序返回事件之前，WaitNextEvent 执行其它进程，并可以截取事件。WaitNextEvent 函数截取 Command-Shift-数字键序列，并调用相应的 'FKEY' 资源来执行有关动作。将 ScrDmpEnable 全局变量（一个字节）设定为0，便可以使事件管理器对数字为 3 － 9 的 Command-Shift-数字键序列中的处理失效。

如果设定了闹钟警报，而且当前时间是报警时间，调用 WaitNextEvent 函数便可以取消报警。用户可以使用闹钟桌面附件设定闹钟警报。

WaitNextEvent 函数还可以调用 SystemTask 过程，为每个已打开的桌面附件或设备驱动程序提供时间，执行为后者定义的一切周期动作。桌面附件或设备驱动程序指定了周期动作发生的频率，SystemTask 可在适当的时间间隔给桌面附件或设备驱动程序提供 CPU 时间。

某些高级事件可以用它们的事件记录来完全指定，而另外一些高级事件可能还需要它们在可选缓冲区中的附加信息。如果需要得到一些附加信息并找到事件的发送程序，请调用 AcceptHighLevelEvent 函数。

如果返回的事件是一个高级事件，而且您的应用程序支持苹果事件，请使用苹果事件管理器的 AEProcessAppleEvent 函数对苹果事件做出响应，并获得得和苹果事件有关的附加信息。

特殊考虑

在 System 7 中，如果您的应用程序跑在前台，而且用户从苹果菜单中打开了一个桌面附件或其它项目，用户就可以单击另一个应用程序或桌面附件的窗口，或者从应用程序菜单中选择另一个应用项目，WaitNextEvent 函数向您的应用程序报告的下一个事件就成为挂起事件。在您的应用程序切换到后台去后，事件管理器会将事件（除了您的应用程序在后台可以接收的事件外）引向新激活的进程，直至用户重又切换回您的应用程序或另一个应用程序为止。

附　注：

事件管理器在 System 6 中的单一任务环境下和在多任务环境下（在多任务环境下，桌面附件是在应用程序分区中启用的。例如，用户从苹果菜单中打开桌面附件时，需同时按下 Option 键）处理桌面附件事件的方式稍有不同。

在这些环境中，当鼠标释放、激活、更新和键盘事件（包括和菜单命令等效的键盘动作）发生时，事件管理器检查现有窗口是否属于桌面附件，以及桌面附件是否能够处理该事件。如果能够，它将事件发送到桌面附件，同时 WaitNextEvent 函数会给您的应用程序返回 FALSE。还要注意，在这些环境中，无论鼠标按下事件是否是针对一个桌面附件生成的，事件管理器对鼠标按下事件总是返回 TRUE。 对于这些情况下鼠标按下事件，如果鼠标按钮是在光标处在桌面附件窗口时按下的（FindWindow 函数返回的 inSystem 常数可以说明这一点），那么您的应用程序就应该调用 SystemClick 过程。SystemClick 过程可以对鼠标按下事件进行处理以适合桌面附件，如果桌面附件窗口需要激活，那么还需要向您的应用程序发送一个激活事件来失活它最上面的窗口。

亦请参阅

请参阅 2-23 页的程序清单 2-1 和 2-24 页的程序清单 2-2 中有关使用 WaitNextEvent 函数的实例。

如果要得到有关高级事件发送程序和检索有关该高级事件的任何附加数据的信息，请参阅2-90页中 AcceptHighLevelEvent函数的说明。有关如何处理苹果事件的细节，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中对 AEProcessAppleEvent 函数的介绍。

有关如何检索一个事件，而不将它从事件流中取消的内容，请参阅下面对 EventAvail 函数的介绍。

EventAvail

您可以使用 EventAvail 函数从事件管理器中检索下一个可用的事件，而无需从应用程序的事件流中取消返回的事件。

```
FUNCTION EventAvail (eventMask: Integer;
                     VAR theEvent: EventRecord): Boolean;
```

eventMask这是一个用来表明返回事件类型的值 该参数可以被解释为事件类型屏蔽参数常数的和。您可以使用这些常数来指定事件类型屏蔽参数:

```
CONST
mDownMask      = 2;     {mouse-down event  (bit 1)}
mUpMask        = 4;     {mouse-up event    (bit 2)}
keyDownMask    = 8;     {key-down event    (bit 3)}
keyUpMask      = 16;    {key-up event      (bit 4)}
autoKeyMask    = 32;    {auto-key event    (bit 5)}
updateMask     = 64;    {update event      (bit 6)}
diskMask       = 128;   {disk-inserted event (bit 7)}
activMask      = 256;   {activate event    (bit 8)}
highLevelEventMask
               = 1024;  {high-level event (bit 10)}
osMask         = -32768;{operating-system (bit 15)}
```

如果需要接收所有的事件，您可以将此参数指定为everyEvent常数

```
CONST
everyEvent     = -1;    {every event}
```

如果事件队列中没有指定类型的事件可返回，EventAvail 将返回一个空事件。

theEvent 一个指定类型或多个指定类型的下一个可用事件。EventAvail 函数将返回的事件从事件流中取消，有关该事件的信息通过事件记录返回。事件记录包括收到事件的类型和其它信息。

说明

如果返回的事件是一个空事件，EventAvail 将返回函数结果 FALSE；否则，EventAvail 返回 TRUE。

和 WaitNextEvent 一样，EventAvail 函数可以调用 SystemTask 过程，它能为每个已打开的桌面附件或设备驱动程序提供时间，以执行为后者定义的周期动作。如果设定了闹钟警报，而且当前时间是报警时间，EventAvail 函数还可以取消警报。用户可以使用闹钟桌面附件设定警报。

特殊考虑

如果 EventAvail从操作系统事件队列中返回了一个低级事件，同时用事件队列满员而从队列中删除了该事件，那么以后将不能存取该事件；但是这种情形并不经常发生。

亦请参阅

请参阅 2-79 页的“事件记录”中有关事件记录域的说明。

GetNextEvent

尽管多采用 WaitNextEvent 检索事件，但也可以通过 GetNextEvent 函数从事件管理器中一次检索一个事件。

```
FUNCTION GetNextEvent (eventMask: Integer;
                       VAR theEvent: EventRecord): Boolean;
```

eventMask 是一个用来表明返回事件类型的值；该参数可以解释为事件类型屏蔽参数常数的和（在 2-26 页的“设定事件类型屏蔽参数”中列出）。如果事件队列中没有指定类型的事件可返回，GetNextEvent 则返回一个空事件。

theEvent 一个指定类型或多个指定类型的下一个可用事件。GetNextEvent 函数将返回的事件从事件流中取消，并通过一个事件记录中返回有关该事件的信息。事件记录包括收到事件的类型和其它信息。

说明

如果返回的事件是一个空事件或 GetNextEvent 截取了该事件，GetNextEvent 将返回函数结果 false; 否则，GetNextEvent 返回 TRUE。GetNextEvent 函数可以调用操作系统管理器的函数 SystemEvent 函数来确定事件应该由应用程序处理还是操作系统处理。

和 WaitNextEvent 一样，GetNextEvent 函数可以调用 SystemTask 过程，它可以为每个已打开的桌面附件或设备驱动程序提供时间，以执行为后者定义的周期动作。如果设定了闹钟警报，而且当前时间是报警时间，GetNextEvent 函数也可以取消警报。(用户可以使用闹钟桌面附件设定警报。)

GetNextEvent 函数也截取 Command-Shift-数字键序列，并调用相应的 'FKEY' 资源来执行有关的动作。将 ScrDmpEnable 全局变量（一个字节）设定为 0，便使事件管理器对数字为 3 - 9 的 Command-Shift- 数字键序列的处理失效。

特殊考虑

为了给多任务环境提供更有力的支持，您的应用程序应该尽可能使用 WaitNextEvent，而不使用 GetNextEvent。如果您的应用程序调用的是 GetNextEvent，应相匹配地调用 SystemTask 过程。

亦请参阅

请参阅 2-79 页的“事件记录”中有关事件记录域的说明。请参阅 2-95 页有 SystemTask 的信息。

AcceptHighLevelEvent

在接收了一个高级事件（除了苹果事件外）后，请调用 AcceptHighLevelEvent 函数来获得有关该事件的附加信息。

```
FUNCTION AcceptHighLevelEvent (VAR sender: TargetID;
                               VAR msgRefcon: LongInt;
                               msgBuff: Ptr;
                               VAR msgLen: LongInt): OSErr;
```

`sender` 识别事件的发送程序; 这一信息通过一个目标标识符记录返回。sender参数包含识别和其它应用程序连接的对话参考号，以及发送程序的端口名称和位置名称。

msgRefcon 可以唯一地确定和这个事件有关的通讯。如果您要对该事件响应，您就应该为 msgRefcon 参数指定同样的值，使得事件的发送程序将应答和原来的请求联系起来。

msgBuff 指定 AcceptHighLevelEvent 函数应该将（和该事件有关的）附加信息返回何处方。您的应用程序负责为 msgBuff 参数指向的附加数据分配存储器，并将 msgLen 参数设定为您为数据分配的字节数。

如果 msgBuff 参数指向存储器中的那个区域，容纳不下和事件有关的所有数据，AcceptHighLevelEvent 会将指定存储器区能够容纳的所有数据返回，并在 msgLen 参数中返回余下的数据量以及结果代码 bufferIsSmall。

msgLen 包含 msgBuff 参数指向的数据大小（以字节为单位）。如果 AcceptHighLevel-Event 返回结果代码 bufferIsSmall，msgLen 参数则包含留下的字节数。您可以再次调用 AcceptHighLevelEvent 来接收其余的数据。

说明

当您的应用程序收到一个高级事件时，您可以使用 AcceptHighLevelEvent 函数得到和该事件有关的附加数据。AcceptHighLevelEvent 函数返回识别事件发送程序的信息，以及事件的唯一消息参考常数。

您的应用程序应该为与事件有关的任何附加信息分配存储器，然后提供数据区域的指针，并以字节为单位提供数据区域的长度。

特殊考虑

AcceptHighLevelEvent 函数可能会移动或清除存储器。您不应在中断过程中调用此函数，例如在一个完成例程或 VBL 任务中。

汇编语言信息

AcceptHighLevelEvent 函数的陷阱宏和例程的选择程序是:

陷阱宏	选择程序
_OSDispatch	$0033

结果代码

noErr	0	没有错误
bufferIsSmall	-607	缓冲区太小
noOutstandingHLE	-608	没有未完成的高级事件

亦请参阅

有关如何使用 AEProcessAppleEvent 函数处理苹果事件的细节，请参阅《Inside Macintosh: Interapplication Communication》。

GetSpecificHighLevelEvent

您可以调用 GetSpecificHighLevelEvent 函数从应用程序的高级事件队列中有选择地检索一个特殊的高级事件。

```
FUNCTION GetSpecificHighLevelEvent
                    (aFilter: GetSpecificFilterProcPtr;
                     yourDataPtr: UNIV Ptr; VAR err: OSErr): Boolean;
```

aFilter　指定过滤函数，GetSpecificHighLevelEvent 通过过滤函数来搜索一个特殊事件。GetSpecificHighLevelEvent 搜索应用程序高级事件队列中的每个事件，直至过滤函数返回 TRUE 或到达队尾为止。

youDataPtr　为您的过滤函数指定了搜索条件。例如，在 youDataPtr 参数中，您可以为搜索一个特殊事件指定一个参考常数，为搜索事件的发送程序指定目标标识符记录的指针，或者为搜索特殊类的事件指定一个事件类。

err　GetSpecificHighLevelEvent 通过这个参数返回错误信息，如果没有出现错误，err 参数为 noErr 常数；如果没有高级事件在您的应用程序的高级事件队列中等待处理，err 参数为 noOutstandingHLE。

说明

您可以使用 GetSpecificHighLevelEvent 函数在应用程序的高级事件队列中搜索特殊的高级事件。您需要提供一个过滤函数的指针，作为 GetSpecificHighLevelEvent 的一个参数。GetSpecificHighLevelEvent 可以为应用程序高级事件队列中的每个事件调用一次过滤函数，直至您的过滤函数返回 TRUE 或到达队列的结尾为止。

GetSpecificHighLevelEvent 可以将您在 youDataPtr 参数中指定的值传送给您的过滤函数。您的过滤函数还接收和该高级事件有关的事件记录以及识别事件发送程序的目标标识符记录，把它们作为参数。您的过滤函数可以将 youDataPtr 参数的内容和它收到的任何其它信息进行比较。

如果您的过滤函数找到了一个匹配事件，必要时它可以调用 AcceptHighLevelEvent 获取进一步信息，然后返回 TRUE。如果您的过滤函数未找到匹配事件，它应该返回 FALSE。

如果您的过滤函数返回 TRUE， GetSpecificHighLevelEvent 函数则返回 TRUE。如果您的过滤函数对应用程序高级事件队列中的所有高级事件都返回 FALSE,或者队列中没有高级事件，那么 GetSpecificHighLevelEvent 将返回 FALSE。

特殊考虑

GetSpecificHighLevelEvent 可能会移动或清除存储器。您不应该在中断过程中调用该函数，例如在一个完成例程或 VBL 任务中。

汇编语言信息

GetSpecificHighLevelEvent 函数的陷阱宏和例程的选择程序是:

陷阱宏	选择程序
_OSDispatch	$0045

亦请参阅

有关如何定义过滤函数及 GetSpecificHighLevelEvent 传送递过滤函数的参数，请参阅 2-114 页的“搜索高级事件队列的过滤函数”。

FlushEvents

FlushEvents 可以用来删除操作系统事件队列中的低级事件。注意，FlushEvents 不能取消操作系统事件队列之外的事件。

在 Finder 装入您的应用程序时，用户可能会生成按键和鼠标事件，它们将进入操作系统事件队列。当您的应用程序首次清除操作系统事件队列中的这些事件时，您可以选择使用 FlushEvents 过程。但是，一般来说，您的应用程序不应该在其他时间清除队列，因为这会导致用户动作丢失，和应用程序及电脑对用户响应失效。

```
PROCEDURE FlushEvents    (whichMask: Integer; stopMask: Integer);
```

whichMask　可以指定哪些种类的低级事件应从操作系统事件队列中删除的参数。这个参数可以解释为事件类型屏蔽参数常数的和。whichMask 和 stopMask 参数共同指定了需要删除的低级事件。

stopMask 是限制从操作系统事件队列中删除事件的参数；这个参数可以解释为事件类型屏蔽参数常数的和。FlushEvents 不取消由 stopMask 参数指定的低级事件。如果要删除由 whichMask 指定的事件，则应将 stopMask 参数指定为 0。

说明

FlushEvents 只删除操作系统事件队列中储存的低级事件；而不删除激活、更新、操作系统或高级事件。

您可以用 whichMask 和 stopMask 参数来指定哪些低级事件应该删除。FlushEvents 可以删除由 whichMask 参数指定的低级事件，直至（但不包括）stopMask 参数指定的第一个类型的事件为止。

如果事件队列不包含由 whichMask 参数指定的事件，那么 FlushEvents 不从队列中删除任何事件。

汇编语言信息

您必须在调用 FlushEvents 前，用事件类型屏蔽参数（whichMask）和停止掩码设置寄存器 D0。当 FlushEvents 返回时，如果所有事件都从队列中删除了，寄存器 D0 为 0；或者，如果所有事件没有从队列中删除，指定事件类型的事件代码使删除进程停止。

寄存器在输入时

D0　　事件类型屏蔽参数（低位字）

　　　停止屏蔽参数（高位字）

寄存器在输出时

D0　　如果所有事件都从队列中删除该寄存器值，为 0；否则为停止搜索事件的事件代码（低位字）

亦请参阅

有关如何指定一个事件类型屏蔽参数的信息，请参阅 2-26 页的“设定事件类型屏蔽参数”。

SystemClick

在收到一个鼠标按下事件后，您的应用程序应该调用窗口管理器的 FindWindow 函数来确定按下鼠标按钮时光标所在位置。如果 FindWindow 返回 isSysWindow 常数（即表明点按在桌面上），请调用 SystemClick 来处理该事件。

```
PROCEDURE SystemClick  (theEvent: EventRecord;
                        theWindow: WindowPtr);
```

theEvent　该事件的事件记录。

theWindow　鼠标按下事件发生的窗口。将FindWindow返回的窗口指针传递给这个参数。

说明

如果一个桌面附件窗口中发生了一个鼠标按下事件，SystemClick 过程可以确定当按下鼠标按钮时，光标在桌面附件窗口的哪一部分，并在必要时将事件传递适当的桌面附件。

如果鼠标按钮按下时，光标在桌面附件窗口的内容区域，而且窗口是激活的，那么 SystemClick 就可以将鼠标按下事件送到桌面附件去处理。如果鼠标按下事件发生在窗口的内容区域，而窗口是非当前活动窗口，SystemClick 则可以使它成为现有窗口。SystemClick 可以通过向您的应用程序发送一个激活事件来失活其最上面的窗口，并将事件导向桌面附件来激活其窗口。

如果鼠标按钮按下时，光标在拖拽区域或转换区域，SystemClick 就会根据需要调用适当的窗口管理器例程 DragWindow 或 TrackGoAway。如果 TrackGoAway 报告说，用户关闭了桌面附件，SystemClick 就会向桌面附件发送一个关闭消息。

亦请参阅

有关事件记录中域的说明，请参阅 2-79 页的“事件记录”。

SystemTask

在协同多任务环境下，WaitNextEvent 函数负责为每个已打开的桌面附件或驱动程序安排执行周期动作的时间。如果您的应用程序调用了 WaitNextEvent，您就不应该再调用 SystemTask。

如果您的应用程序调用 GetNextEvent，它应该再调用 SystemTask 过程。

```
PROCEDURE SystemTask;
```

说明

SystemTask 过程可以为每个已打开的桌面附件或驱动程序安排执行周期动作的时间，该动作是专为桌面附件或驱动程序而定义的。一个桌面附件或设备驱动程序可以指定周期动作发生的频率，SystemTask 按适当的间隔给桌面附件或设备驱动程序分配 CPU 时间。

如果您的应用程序调用 GetNextEvent，它至少应该每 1/6 秒调用一次 SystemTask。通常这相当于每通过您的事件循环一次就调用一次。如果您的应用程序进行大量的处理，那么您可能需要在您的事件循环中多次调用 SystemTask。

亦请参阅

有关 WaitNextEvent 函数和 GetNextEvent 函数的说明，请分别参阅 2–85 页和 2–89 页所介绍的内容。

SystemEvent

WaitNextEvent 函数和 GetNextEvent 函数均可调用 SystemEvent 函数。但在大多数情况下，您的应用程序不应该调用 SystemEvent 函数。

SystemEvent 函数可以确定一个特殊的事件应该由应用程序还是操作系统来处理。

```
FUNCTION SystemEvent(theEvent: EventRecord): Boolean;
```

`theEvent` 该事件的事件记录。

说明

如果事件应该由应用程序处理，SystemEvent 返回函数结果 FALSE；否则，SystemEvent 将采取适当的动作并返回 TRUE。

对激活、更新、鼠标释放和键盘事件（包括和命令等效的键盘动作），SystemEvent 检查当前活动窗口是否属于一个桌面附件，以及该桌面附件是否能够处理那类事件。如果能够处理，SystemEvent 则将事件发送给桌面附件并返回 TRUE。否则，SystemEvent 返回 FALSE。

对鼠标按下事件和空事件，SystemEvent 返回 FALSE。

对磁盘插入事件，SystemEvent 通过调用 PBMountVol 函数安装磁盘，但返回值 FALSE，以便应用程序必要时可以执行进一步的处理。

汇编语言信息

如果 SEvtEnb 全局变量（一个字节）为 0，SystemEvent 则总是返回 FALSE。

亦请参阅

有关事件记录中域的说明，请参阅 2–79 页的“事件记录”。有关 PBMountVol 函数的说明，请参阅《Inside Macintosh 文件》(*Inside Macintosh: File*) 一书中的“文件管理器”一章。

GetOSEvent

工具箱事件管理器可以通过调用 GetOSEvent 函数从操作系统事件队列中检索低级事件。但在大多数情况下，您的应用程序不应该使用这个函数。

```
FUNCTION GetOSEvent (mask: Integer;
                     VAR theEvent: EventRecord): Boolean;
```

`mask` 设定返回事件类型的参数; 这个参数可以解释为事件类型屏蔽参数常数的和。GetOSEvent 只返回储存在操作系统事件队列中的低级事件；而不返回激活、更新、操作系统或高级事件。如果没有任何指定类型的低级事件可用，GetOSEvent 将返回一个空事件。

`theEvent` 操作系统事件队列中下一个可用的指定类型事件。GetOSEvent 函数可以从操作系统事件队列中删除返回的事件，并通过一个事件记录中返回有关该事件的信息。事件记录包括接收到的事件类型和其它信息。

说明

GetOSEvent 函数可以从操作系统事件队列中检索并删除一个事件。如果返回的事件是一个空事件，GetOSEvent 返回函数结果 FALSE; 否则，GetOSEvent 返回 TRUE。GetOSEvent不以任何形式截取和响应事件。它也不处理 Command-Shift-数字键组合，不处理用户通过闹钟桌面附件设定的任何警报。

汇编语言信息

您必须在请求 GetOSEvent 函数前，用一个事件记录的地址设置寄存器 A0，用事件掩码设置寄存器 D0。当 GetOSEvent 返回时，寄存器 D0 可以表明返回的事件是空事件还是非空事件，返回的事件可以通过寄存器 A0 存取。

寄存器在输入时

A0	事件记录的地址
D0	事件类型屏蔽参数（低位字）

寄存器在输出时

A0	事件记录的地址
D0	如果 GetOSEvent 返回一个非空事件，为 0；如果返回一个空事件，为 -1（低位字节）

亦请参阅

有关事件记录中域的说明，请参阅 2-79 开始的“事件记录”。有关如何指定事件类型屏蔽参数的信息，请参阅 2-26 页开始的“设定事件类型屏蔽参数”。

OSEventAvail

工具箱事件管理器可以使用 OSEventAvail 函数从操作系统事件队列中检索一个事件，而不删除它。在大多数情况下，您的应用程序不应使用这个函数。

```
FUNCTION OSEventAvail  (mask: Integer;
                         VAR theEvent: EventRecord): Boolean;
```

mask　指示返回事件类型的值；这个参数可以解释为事件类型屏蔽参数常数的和。OSEventAvail 只返回储存在操作系统事件队列中的低级事件；而不返回激活、更新、操作系统或高级事件。如果没有任何指定类型的低级事件可用，OSEventAvail 将返回一个空事件。

theEvent　操作系统事件队列中下一个可用的指定类型事件。OSEventAvail 函数不会从操作系统事件队列中删除返回的事件，它通过一个事件记录中返回有关该事件的信息。事件记录包括收到的事件类型和其它信息。

说明

OSEventAvail 函数可以从操作系统事件队列中检索一个事件，而不删除它。如果返回的事件是一个空事件，OSEventAvail 函数结果返回 FALSE；否则，OSEventAvail 将返回 TRUE。

OSEventAvail 不以任何形式截取和响应事件。同时也不处理 Command-Shift-数字键组合，不处理用户通过闹钟桌面附件设定的任何警报。

特殊考虑

在操作系统事件队列中，如果 OSEventAvail 函数返回了一个低级事件，但事件队列已满，于是便将该事件从队列中删除，那么以后该事件将不能存取；但是，这并不经常发生。

汇编语言信息

您必须在请求 OSEventAvail 函数前，用一个事件记录的地址设置寄存器 A0，用事件掩码设置寄存器 D0。当 OSEventAvail 返回时，寄存器 D0 表明返回的事件是空事件还是某种其它事件，返回的事件可以通过寄存器 A0 存取。

寄存器在输入时

A0　　事件记录的地址
D0　　事件类型屏蔽参数（低位字）

寄存器在输出时

A0　　事件记录的地址
D0　　如果 OSEventAvail 返回一个非空事件，该寄存器值为 0；如果返回一个空事件，为 -1（低位字节）

亦请参阅

有关事件记录中域的说明，请参阅 2-79 的“事件记录”。有关如何指定事件类型屏蔽参数的信息，请参阅 2-26 页的“设定事件类型屏蔽参数”。

SetEventMask

SetEventMask 过程可以将应用程序的事件类型屏蔽参数设定成指定的屏蔽参数。您的应用程序不应该调用 SetEventMask 过程来禁止任何事件类型的传递。只有在您的应用程序需要对键释放事件响应时，SetEventMask 才可用来启用键释放事件。

```
PROCEDURE SetEventMask (theMask: Integer);
```

`theMask`　　指定能够向操作系统事件队列传递的事件类型的事件类型屏蔽参数

说明

SetEventMask 过程可以根据参数 theMask 设定应用程序的事件类型屏蔽参数。当向操作系统事件队列传递事件时，操作系统事件管理器只传递低级事件（除了更新或激活事件），这些低级事件相应于当前进程的系统事件类型屏蔽参数中的位。应用程序的系统事件类型屏蔽参数的初始设定是向操作系统事件队列传递释放鼠标、按下鼠标、键按下、自动键和磁盘插入事件。

汇编语言信息

当前应用程序的系统事件类型屏蔽参数可在 SetEventMask 系统全局变量中获得。

亦请参阅

有关事件类型屏蔽参数的附加信息，请参阅 2-26 页的“设定事件类型屏蔽参数”。

GetEvQHdr

事件管理器可以通过调用 GetEvQHdr 来得到操作系统事件队列头指针。在大多数情况下，您的应用程序不应该调用 GetEvQHdr 函数。

说明

GetEvQHdr 函数返回操作系统事件队列头指针。

汇编语言信息

EventQueue 系统全局变量包含事件队列头指针。

亦请参阅

有关操作系统事件队列结构的信息，请参阅 2-83 页的“事件队列”。

发送事件

您可以通过 PostHighLevelEvent 函数向其他应用程序或进程发送事件。如果要向其他应用程序发送苹果事件，请调用苹果事件管理器 AESend 函数。操作系统事件管理器还提供 PPostEvent 和 PostEvent 函数来向操作系统事件队列传钉低级事件。工具箱事件管理器可以使用 PostEvent 函数，但在大多数情况下，您的应用程序不应该使用 PostEvent 函数。

PostHighLevelEvent

您可以通过 PostHighLevel 函数向另一应用程序发送高级事件。

```
FUNCTION PostHighLevelEvent  (theEvent : EventRecord ;
                              receiverID : Ptr ; msgRefcon : LongInt ;
                              msgBuff : Ptr ; msgLen : LongInt ;
                              postingOptions : LongInt) : OSErr ;
```

theEvent　发送的事件。您的应用程序应该填写事件记录的 what，message 和 where 字段。在 what 域中指定 kHighLevelEvent 常数，在 message 域中指定高级事件的事件类，在 where 域中指定事件标识符。您不必填写 when 和 modifiers 域；当您发送该消息时，事件管理器会自动给这些域分配正确的值。

receiverID　高级事件接收者。当在本地机上向另一应用程序发送事件时，您可以通过对话参考号码，进程序号，署名，或端口名称和位置名称来指定事件的接收者。在向远程机中的应用程序发送事件时，您可以通过对话参考号或端口名称和位置名称来指定接收者。

如果需要指定端口名称和位置名称，可在 receiverID 参数中提供一个目标标识符记录的地址。如果希望指定一个进程序号，则可以在 receiverID 参数中提供它的地址。指定对话参考号，或署名，可以在 receiverID 参数中提供数据　应为 Ptr 数据类型)。

msgRefcon　识别与该事件相关联的通讯信息的唯一号码。您的应用程序可以把该域设定为它选择的任意值。如果您正在应答一个高级事件，您应该在 msgRefoon参数中使用与产生该请求的高级事件中所指定的同样的值。

msgBuff　一个数据缓冲区指针，该缓冲区包含关于该事件的一些附加信息。

msgLen　由 msgBuff 所指的数据缓冲区的大小　以字节计算)。

postingOptions

与 receiverID 参数相关联的选项和与事件相关联的发送选项。您可以指定一个或多个发送选项以表明您是否想要另一应用程序在下次有机会时接收该事件，是否想要事件被另一应用程序接收的回执。将可以使用与receiverID参数相关联的选项来表明您是怎样指定该事件的接收者的—是通过目标标识符记录中的端口名称和位置名称，还是通过对话参考号、进程序号或署名。

您可以在 postingOptions 参数中把这些常数组合起来使用

```
CONST
nAttnMsg
     = $00000001; {give this message priority}
nReturnReceipt
     = $00000200; {return receipt requested}
receiverIDisTargetID
     = $00005000; {ID is port name and location name}
receiverIDisSessionID
     = $00006000; {ID is PPC session reference number}
receiverIDisSignature
     = $00007000; {ID is creator signature}
receiverIDisPSN
     = $00008000; {ID is process serial number}
```

说明

PostHighLevelEvent 函数将高级事件发送给指定的进程。

如果您正在向一个应用程序发送高级事件时该应用程序终止了，则当您下次调用 PostHighLevelEvent 向终止的应用程序发送另一高级事件时，您将接收到结果码 sessionClosedErr。如果您不关心有关该对话的任何状态信息，您就可重新发送您的事件。否则您必须重新启动另一对话并重新发送您的事件。

如果您的应用程序正在后台运行并发送一个要求显示网络通讯对话框的高级事件时 PostHighLevelEvent 返回 noUserInteractionAllowed 结果代码，而不显示网络通讯对话框，也不发送该事件。如果您的应用程序接收到 noUserInteractionAllowed 结果码，您可以使用通知管理器来通知用户您的应用程序需要引起注意。当用户将您的应用程序带到前台时，您可以报告该事件。如果重新发送获得成功，您的应用程序可以继续发送高级事件而不需进行与用户的进一步对话。注意，PostHighLevelEvent 仅在第一次向远程目标发送高级事件时能返回 noUserInteractionAllowed。

特殊考虑:

PostHighLevelEvent 函数可能会移动或清除存储器。您不应在中断过程中（诸如完成（Completion 例程或 VBL 任务）调用此函数。

汇编语言信息

PostHighLevelEvent 函数的陷阱宏和例程选择程序是:

陷阱宏	**选择程序**
_OSDispatch	$ 0034

亦请参阅

若想了解关于怎样使用 AESend 函数发送苹果事件的详细情况，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书。

结果代码

noErr	0	No error
connectionInvalid	−609	Connection is invalid
noUserInteractionAllowed	−610	Cannot interact directly with user
sessionClosedErr	−917	Session closed

PPostEvent

大多数情况下您的应用程序不必在操作系统事件队列中发送事件; 但如果您必须这样做，您可以使用 PPostEvent 函数。

```
FUNCTION PPostEvent (eventCode: Integer; eventMsg: LongInt;
                     VAR qEl: EvQElPtr): OSErr;
```

eventCode　一个表示传递到操作系统事件队列中的事件类型数值。mouseDown、mouseUp, keyDown, keyUp, autokey 和 diskEvt 这些常数代表可在该队列中被传递的事件类型。不要试图在操作系统事件队列中传递任何其它事件类型。

eventMsg　是一个长整数，它包含 PPostEvent 在队列中应传递事件的 message 域的内容。

qE1　PPostEvent在该参数中返回一个指向被传递事件的事件队列的入口地址指针。

说明

在eventCode和eventMsg参数中，您可以指定该事件的事件记录中的what和message域值。PPostEvent函数把当前时间，当前鼠标位置，以及修正键和鼠标按钮的当前状态填写到事件记录的when，where，和modifiers域中。

PPostEvent在qE1参数中返回一个指向被投寄事件的事件队列入口地址的指针。您可以通过改变被投寄事件的事件队列入口地址来改变被传递事件的任意域。例如，您可以通过改变事件队列输入的 evtQModifiers 域的值来改变被发送事件的修正键。

PPostEvent函数仅发送那些由系统事件类型屏蔽参数启用的事件。如果事件队列已满，PostEvent将删除队列中最旧的事件并发送新事件。

▲ 警告

不要在操作系统事件队列中发送除鼠标按下、鼠标释放、键按下、键释放、自动键和磁盘插入事件以外的任何事件。试图将其它事件发送到操作系统事件队列中将会影响事件管理器的内部工作。

汇编语言信息

您必须在调用 PPostEvent 之前设定寄存器 A0 和寄存器 D0。PPostEvent 函数的返回值存于寄存器 A0 和 D0 中。

寄存器在输入时

A0	事件号码	低位字）
D0	事件消息	长型）

寄存器在退出时

A0	指向一事件队列输入的指针	长型）
D0	结果代码	低位字）

结果码

evtNotEnb	1	事件类型无效—事件没被发送。
noErr	0	无错误

亦请参阅

若想了解关于事件队列中入口地址的说明，请参阅第 2-83 页的“事件队列”。

PostEvent

工具箱事件管理器可以使用 PostEvent 函数将事件发送到操作系统事件队列中。在大多数情况下，您的应用程序不应调用 PostEvent 函数。

```
FUNCTION PostEvent (eventNum: Integer; eventMsg: LongInt): OSErr;
```

eventNum　表示发送到操作系统事件队列中事件类型的数值。mouseDown、mouseUp、keyDown、keyUp、autoKey 和 diskEvt 这些常数代表那些可在该队列中投寄的事件类型。不要试图在操作系统事件队列中发送任何其它事件类型。

eventMsg　是一个长整数，它包含 PostEvent 应在队列中发送的事件 message 域的内容。

说明

在 eventNum 和 eventMsg 参数中，您可以为本事件的事件记录中的 what 和 message 域指定一个值。PostEvent 函数则能够把当前时间、当前鼠标位置以及修正键和鼠标按钮的当前状态填写到事件记录的 when、where 和 modifiers 域中。

PostEvent函数仅发送那些由系统事件类型屏蔽参数启用的事件。如果事件队列已满，PostEvent 将删除队列中最陈旧的事件并发送新事件。

注意，如果您使用 PostEvent 函数来重新发送事件，PostEvent 函数将填写事件记录的 when， where，和 modifier 域，给这些域赋予重发事件值，但这些值与原来事件中的值不同。

▲ 警告
不要在操作系统事件队列中发送除按下鼠标、释放鼠标、键按下、键释放、自动键和磁盘插入事件以外的任何事件。如果试图将其它事件发送到操作系统事件队列中将会影响事件管理器的内部工作。

汇编语言信息

您必须在调用 PostEvent 之前将寄存器 A0 设置的事件代码，将寄存器 D0 设置为事件信息。PostEvent 返回时，寄存器 D0 含有结果码。

寄存器在输入时

A0	事件号码（低位字）
D0	事件消息（长型）

寄存器在退出时

D0	结果代码（低位字）

结果代码

evtNotEnb	1	事件类型无效—事件未被发送。
noErr	0	无错误

转换进程序号和端口名称

事件管理器可以提供两个函数来实现进程序号和端口名称间的转换（GetProcessSerial-NumberFromPortName）和 GetPortNameFromProcessSerialNumber）。这两个函数均可用来将序号映像到端口名称（反之亦然），以便在本地机上打开应用程序。它们并不会为远程机上打开的应用程序返回有用的结果。

GetProcessSerialNumberFromPortName

使用 GetProcessSerialNumberFromPortName 获取一个进程序号。

```
FUNCTION GetProcessSerialNumberFromPortName
            (portName: PPCPortRec;
            VAR PSN: ProcessSerialNumber): OSErr;
```

portName　一个进程的端口名称，该进程序号是您所希望选择的那一种。

PSN　返回由portName参数指定的进程的进程序号。您可以使用返回的进程序号向该进程发送高级事件。无需解释进程序号的值。

说明

GetProcessSerialNumberFromPortName 函数返回在一特定端口登记的进程序号。

特殊考虑

GetProcessSerialNumberFromPortName 函数并不移动或清除内存块，但由于其它原因不应在中断过程中（诸如在完成例程或 VBL 任务中）调用该函数。

汇编语言信息

GetProcessSerialNumberFromPortName 函数的陷阱宏和例程的选择程序是:

陷阱宏	选择程序
_OSDispatch	$ 0035

结果代码

noErr	0	无错误
noPortErr	–903	无效端口名称

亦请参阅

若想了解关于PPCPortRec数据类型的说明，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中“程序间通信”一章。

GetProtNameFromPorcessSerialNumber

使用 GetProtNameFromPorcessSerialNumber 获取一个进程的端口名称。

```
FUNCTION GetPortNameFromProcessSerialNumber
            (VAR portName: PPCPortRec;
            PSN: ProcessSerialNumber): OSErr;
```

`portName` 返回由 PSN 参数指定的进程端口名称。您可以使用返回的端口名称向该进程发送高级事件。

`PSN` 您想得到其端口名称的进程序号。

说明

GetPortNameFromProcessSerialNumber函数返回的是带有特定进程序号的进程所登记的端口名称。

特殊考虑

GetPort NameFromProcessSerialNumber 函数并不移动或清除内存块，但由于其它原因，不应在中断过程中，诸如在一完成例程或 VBL 任务中，调用该函数。

汇编语言信息

GetProtNameFromPorcessSerialNumber 函数的陷阱宏和例程选择程序是

陷阱宏	**选择程序**
`_OSDispatch`	$0046

结果代码

noErr	0	无错误
procNotFound	−600	没有符合指定的进程序号的进程。

亦请参阅

若想了解关于PPCPortRec数据类型的说明，请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中“程序间通信工具箱”一章。

读鼠标

事件管理器为您提供了可以用来获取鼠标信息的例程。您可以调用 GetMouse 过程获取当前的鼠标位置。您还可以用 button 函数来确定用户是否按下鼠标按钮。在接收到一个鼠标按下事件后，您可以用stillDown函数来确定鼠标按钮是否还处于按下状态，您还可以用 WaitMouseUp 函数来确定用户是否紧接着释放了鼠标。

GetMouse

您可以使用 GetMouse 过程来获取当前鼠标位置。

```
PROCEDURE GetMouse (VAR mouseLoc: Point);
```

`mouseLOC` 返回当前鼠标位置（该位置采用当前图形端口的局部坐标系统）。注意，该数值与事件记录的 where 域的数值不同，where 域中的数值指定了全局坐标中的鼠标位置。

Button

您可以使用 Button 函数来确定用户是否按下鼠标按钮。

```
FUNCT1ON Button: Boolean;
```

说明

Button 函数可以在操作系统事件队列中寻找一鼠标按下事件。若找到，Button 函数将返回 TRUE；否则，它返回 FALSE。在一个鼠标按下事件发生后，若想确定鼠标是否还处于按下状态，可以使用 StillDown 函数。

亦请参阅

请参阅 2-83 “事件队列” 中关于操作系统事件队列的说明信息。

StillDown

在接收到一个鼠标按下事件后，您可以使用 StillDown 函数来确定鼠标按钮是否还处于按下状态。

```
FUNCTION StillDown: Boolean;
```

说明

StillDown 函数可以在操作系统事件队列中寻找鼠标事件。如果找到一个鼠标事件，StillDown 函数则返回 FALSE。如果它在操作系统事件队列中没有找到任何等待处理的鼠标事件，则 StillDown 函数返回 TRUE。

亦请参阅

请参阅 2 - 83 页“事件队列”关于操作系统事件队列的说明信息。

WaitMouseUp

在接收到一个鼠标按下事件之后，您可以通过 WaitMouseUp 函数来确定用户是否紧接着释放了鼠标。

```
FUNCTION WaitMouseUp: Boolean;
```

说明

WaitMouseUp 函数可以在操作系统事件队列中寻找释放鼠标事件。如果它找到一个释放鼠标事件，WaitMouseUp 函数将在队列中删除释放鼠标事件并返回 TRUE。如果它在操作系统事件队列中没有找到任何等待处理的鼠标事件，则 WaitMouseUp 函数返回 FALSE。

亦请参阅

请参阅 2 - 83 页“事件队列”中关于操作系统事件队列的说明信息。

读键盘

当您使用 WaitNextEvent, EventAvail, 或 GetNextEvent 函数向事件管理器发出事件请求时，事件管理器接到您的应用程序的请求后，每次报告一个键盘事件。除用户按下或释放一个键时您可以获取键盘事件以外，您还可以随时调用 GetKeys 过程直接读键盘（和小键盘)。

另外，您还可以调用 KeyTranslate 函数，通过使用一个指定的'KCHR'资源，将模拟键代码转换为字符代码值。

GetKeys

您可以使用 GetKeys 过程来获取键盘的当前状态。

```
PROCEDURE GetKeys (VAR theKeys: KeyMap);
```

theKeys　回车键盘的当前状态，如果有数字小键盘则包括数字小键盘。GetKey 过程使用 KeyMap 数据类型返回该信息。

```
TYPE KeyMap=PACKED ARRAY [0...127] OF Boolean;
```

键盘或数字小键盘上的每一个键对应 KeyMap 阵列中的一个单元。一个特定键的变址等于该键的模拟键代码减 1。例如，模拟键代码为 38 的键（在苹果键盘 II 中的“J”键)，对应于返回的阵列中的 KeyMap [37]，可以通过存取 KeyMap [37] 来存取该键。如果 KeyMap 单元对应的键被按下，其值为TRUE，否则其值为FALSE。可同时按下的键的最大数目为两个字符键加 5 个修正键的任意组合。

说明

您可以使用GetKeys过程来确定键盘的当前状态。例如，您可以使用GetKeys过程来确定一个修正键是被单独按下的还是与另一个键组合起来按下的。

KeyTranslate

您可以使用 KeyTranslate 函数将一个模拟键代码转换为一个基于 'KCHR' 资源的字符代码。KeyTranslate 函数也可以用作 KeyTrans 函数。

```
FUNCTION KeyTranslate (transData:ptr,Keycode:Integer;
                         VAR state: Longlnt):Longlnt;
```

transData　　一个‘KCHR’资源指针。在将键代码转换成一个字符代码时，您希望 KeyTranslate 函数使用该指针。

Keycode　　这是一个您的应用程序应该设定的16位数值，目的是要让它的 0-6 位包含模拟键代码，第7位包含键状态，值为1则表明键释放；为0表明键按下。第8-15位的解释与事件记录的modifier域的高位字节相同，应该根据您应用程序的需要来设定这些位。

State　　状态值。您的应用程序第一次调用 KeyTranslate 或每次使用一个不同的`KCHR’资源调用 KeyTranslate 时都应将该值设定为0。此后，您的应用程序应把前一次调用 KeyTranslate 返回的值传到 State 参数中。

说明

KeyTranslate 函数可以返回一个32位的值，这个值可以给模拟键代码一个字符代码，而这个模拟键代码是由 Keycode 参数指定的。图2－17显示了 KeyTranslate 返回的32位数的结构。

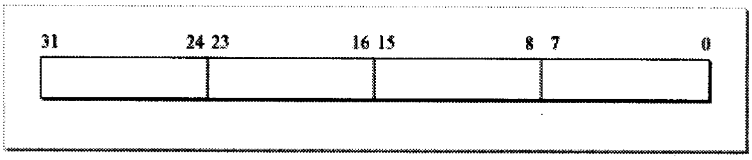


图2-17　KeyTranslate 函数结果的结构

KeyTranslate 函数可以返回对应于一个或两个（可能的话）字符的值，由给定模拟键代码生成。例如，一个给定模拟键代码可能对应着一个带有连字符“－”的字母字符。例如，当用户按下 Option-E（选项-E）之后按N，您就可以使用 U.S. `KCHR’资源通过 KeyTranslate 函数产生 'n 来映象这一事件，KeyTranslate 把 'n 当作两个字符返回标以字符代码1和字符代码2的字节中。如果 KeyTranslate 仅返回了一个字符代码，那么一定是在标以字符代码2的字节中。不管怎样，您的应用程序应该经常检查在图 2－17 中标以字符代码1和字符代码2的两个字节，以获取映像到模拟键代码可能出现的值。

亦请参阅

如果希望进一步了解关于 `KCHR’ 资源和 KeyTranslate 函数的详情，请参阅《Inside Macintosh：Text》一书。

获取定时信息

您可以使用TickCount函数来获得自系统最近一次启动以来的当前Tick数。您还可以使用该函数来比较一个给定的事件或其它动作发生以来经历的Tick 数。

通过使用GetDblTime函数，您可以获得把两个鼠标事件当作一个双击事件建议的 Tick 差上限。用户可以使用鼠标控制屏面来调整该数值。通过使用GetCartTime函数，您可以获得在可编辑文字中的插入光标建议闪烁的最大时间Tick 差。用户可以使用通用控制板调整该参数。

TickCount

您可以使用TickCount函数来获得系统自最近一次启动以来的当前时间Tick数（一Tick大约为1/60秒）。

```
FUNCTION TickCount:Longlnt;
```

说明

TickCount 函数返回一个长整数，该整数可以指示系统自最近一次启动以来的当前Tick 数。您可以使用该数值来对照一个给定事件或其它动作自发生以来经历的Tick数。例如，您可以将由 TickCount 返回的当前数值与事件记录的 when 域值相比较。

Tick 计数将在垂直回扫中断期间获得增量，但也可使该中断失效。您的应用程序不应依赖Tick 计数而能绝对精确地增加Tick 数；您的应用程序也不应假定Tick计数总是以1为增量；一个中断任务可能保持多于一个Tick 的控制。如果您的应用程序记录着先前的Tick 计数并把刻数值与当前Tick 计数相比较，您的应用程序应该通过比较两数值来检验一种“大于或等于”条件而不是“等于先前的Tick 计数加1”。

▲ 警告

Tick计数未必精确；通常它只精确到一个Tick ，但这种精确度也没有保证。

汇编语言附注

TickCount返回的值也可在全局变量Ticks中存取。

GetDblTime

如果需要确定鼠标事件序列是否构成了一个双击事件，您的应用程序可以测量释放鼠标事件和鼠标按下事件之间的时间间隔（以Tick为单位）。如果两个鼠标事件之间的时间间隔小于GetDblTime的返回值，那么您的应用程序就应将这两个鼠标事件解释为一个双击事件。

```
FUNCTION GetDblTime: Longlnt;
```

说明

GetDblTime 函数可以返回释放鼠标事件和鼠标按下事件之间建议的最大时间间隔（以 Tick 为单位）。用户可以用鼠标控制板调整该值。

如果您的应用程序要区分鼠标双击事件和点按事件，则应该用 GetDblTime 返回的值来鉴别。如果您的应用程序使用 TextEdit，那么 TextEdit 过程可以通过适当地醒目显示选定项或取消选定项的醒目显示等方法来自动识别和处理 TextEdit 编辑记录内的文字双击事件。

汇编语言附注

在系统全局变量 DoubleTime 中您也可以存取 GetDblTime 的返回值。

GetCaretTime

您可以使用 GetCaretTime 函数来取得可编辑文字插入光标（通常用一个垂直条来标示插入点）闪烁期间的建议 Tick 差。用户可以使用通用控制板来调整该值。

```
FUNCTION Get CaretTime:Longlnt;
```

说明

如果您的应用程序支持可编辑文字，您的应用程序就应该使用 GetCaretTime 返回的值来确定应用程序闪烁插入光标的频率。如果您的应用程序仅使用 TextEdit，您可以使用 TextEdit 过程自动地闪烁插入光标，其时间间隔为用户在通用控制板中指定的间隔。

汇编语言附注

在系统全局变量 CaretTime 中，您也可以存取 GetCaretTime 返回的值。

应用程序定义的例程

在使用 GetSpecificHighLevelEvent 时，您可以提供一个过滤函数，以便您的应用程序能在高级事件队列中搜索一个特定事件。

搜索高级事件队列的过滤函数

本节说明了您可以提供给 GetSpecificHightLevelEvent 的过滤函数。例如，您可以使用过滤函数来搜索发自一个特定应用程序的高级事件。

用于搜索高级事件队列的过滤函数

当您使用 GetSpecificHighLevelEvent 在应用程序的高级事件队列中搜索一个特定事件时，您可以提供一个指向过滤函数的指针。对于高级事件队列中的每个事件，GetSpecificHighLeveEvent 调用您的过滤函数一次，直到您的过滤函数返回 TRUE 或事件队列结束为止。您的过滤函数可检查每个事件并确定该事件是否是想要的事件。如果是，您的过滤函数就返回 TRUE。

下面告诉您怎样说明过滤函数 MyFilter:

```
FUNCTION MyFilter (YourDataptr: Ptr;
                   msgBuff: HighLevelEventMsgPtr;
                   sender:TargetID): Boolean;
```

`yourDataPtr` 为您的过滤函数获取特定事件指定了一个标准。例如，您可以将 yourDataPtr 参数指定为一个参考常数以搜索特殊事件，指定为一个指向目标标识符 记录的指针以搜索事件的特定发送程序，或者指定为一个事件类以搜索特定的事件类.

`msgBuff` 包含指向数据类型 HighLevelEventMsg 记录的一个指针，该数据类型可以提供: 高级事件的事件记录和事件的参考常数。在 (2-82) 页的“高级事件消息记录”中有关于 HighLevelEventMsg 数据类型的说明。

`sender` 包含发送该事件的应用程序的目标标识符 记录。在 (2-81) 的“目标标识符记录”中有关于 TargetID 数据类型的说明。

说明

过滤函数可以将 yourDataPtr 参数的内容与 msgBuff 和 Sender 参数的内容相比较。如果您的过滤函数找到了一个匹配项，它可以必要时调用 AcceptHighLevelEvent 并返回 TRUE。如果您的过滤函数并没有找到一个匹配项，则应返回 FALSE。

亦请参阅

若想了解怎样为GetSpecificHighLeveEvent函数指定过滤函数的说明，请参阅（2-92）页。

资源

本节解释'SIZE'资源的结构及其每个域的含义。您应该负责创建这个资源中的信息。

'SIZE' 资源

在 System 7 中执行的每个应用程序，以及在 MultiFinder 下执行的每个应用程序，均应包含一个 'SIZE' 资源。'SIZE'资源的主要功能之一就是告诉操作系统应用程序需要的存储器大小，以便操作系统能为该应用程序设置一个大小合适的存储器分区。'SIZE' 资源也被用来向操作系统指示某些调度选项，例如应用程序是否可以接受挂起或恢复执行事件。在 System 7 中的'SIZE'资源中包含的附加信息还可指示应用程序是否是32-位归零，它是否支持静态文稿，它是否使用 TextEdit 的直接插入输入光标，应用程序是否希望接收它启用的任何应用程序的终止通知，以及应用程序是否希望接收高级事件。

'SIZE' 由一个16-位标志域及其后面的两个32-位大小域组成。标志域可以指定应用程序的操作特性，而大小域可以指示应用程序的最小存储器分区和优选存储器分区的大小。最小存储器分区大小是一个实际界限。低于这个存储器大小，您的应用程序将无法运行。优选存储器分区大小是使您的应用程序能够最有效运行的存储器大小，操作系统在这种情况下还要试图保障可靠地启用该应用程序。如果不能得到这样大小的存储器总量，应用程序将被放置在可用的最大连续存储器块中，前提是该存储器块大于指定的最小存储器尺寸。

附　注:

如果可用的存储器总量处在最小和优选的大小之间，Finder将显示一个对话框，询问用户是否要使用可用的存储器总量来运行应用程序。如果您的应用程序没有'SIZE'资源，则给它分配的系统设定分区大小为 512KB，而进程管理器则在所有定义项（通常由标志域中的常数定义）中使用一个 FALSE 系统设定值。

在定义'SIZE'资源时，您应该把－1当作它的资源标识符。用户可以在应用Finder的信息窗口中修改优选的分区大小。如果用户已更改了优选的分区大小，操作系统会创建一个新的'SIZE' 资源，其资源标识符 为0。当用户修改资源中的任何其它设定时，进程管理器也会创建一个新的'SIZE'资源。

在系统软件 7.1 版本中，用户也可以在您的应用 Finder 信息窗口中修改最小存储器分区的大小。在 7.1 版本中，如果用户改变最小或优选的存储器分区大小，作系统将创建两个新的 'SIZE' 资源，一个资源标识符为 0，另一个资源标识符为 1。

在应用程序启用时，进程管理器为了确定优选的存储器分区大小而寻找资源标识符 为 0 的 'SIZE' 资源；如果没有找到该资源，它将使用资源标识符 为 - 1 的初始 'SIZE' 资源。在 7.1 版本中，进程管理器可以寻找一个资源标识符 为 0 的 'SIZE' 资源，用以确定优选的存储器分区大小，寻找一个资源标识符 为 1 的 'SIZE' 资源用以确定最小的存储器分区大小；如果没有找到这些资源，它将使用资源标识符 为 - 1 的初始 'SIZE' 资源。

程序清单 2 - 19 是 Rez 格式的 'SIZE' 资源的结构。请参阅（2 - 30）页开始的“创建一个 'SIZE' 资源”中应用程序 'SIZE' 资源的实例。

程序清单 2 - 19　'SIZE' 资源的 Rez 属性单元

```
type 'SIZE' {
    boolean  reserved;                   /*reserved*/
    boolean  ignoreSuspendResumeEvents,  /*ignores suspend-resume events*/
             acceptSuspendResumeEvents;  /*accepts suspend-resume events*/
    boolean  reserved;                   /*reserved*/
    boolean  cannotBackground,           /*can't use background null events*/
             canBackground;              /*can use background null events*/
    boolean  needsActivateOnFGSwitch,    /*needs activate event following */
                                         /* major switch*/
             doesActivateOnFGSwitch;     /*activates own windows in */
                                         /* response to OS events*/
    boolean  backgroundAndForeground,    /*app has a user interface*/
             onlyBackground;             /*app has no user interface*/
    boolean  dontGetFrontClicks,         /*don return mouse events */
                                         /* in front window on resume*/
             getFrontClicks;             /*do return mouse events */
                                         /* in front window on resume*/
    boolean  ignoreAppDiedEvents,        /*applications use this*/
             acceptAppDiedEvents;        /*app launchers use this*/
    boolean  not32BitCompatible,         /*works with 24-bit addr*/
             is32BitCompatible;          /*works with 24- or 32-bit addr*/
    boolean  notHighLevelEventAware,     /*can't use high-level events*/
             isHighLevelEventAware;      /*can use high-level events*/
    boolean  onlyLocalHLEvents,          /*only local high-level events*/
             localAndRemoteHLEvents;     /*also remote high-level events*/
    boolean  notStationeryAware,         /*can't use stationery documents*/
             isStationeryAware;          /*can use stationery documents*/
    boolean  dontUseTextEditServices,    /*can't use inline services*/
             useTextEditServices;        /*can use inline services*/
```

```
    boolean   reserved;                         /*reserved*/
    boolean   reserved;                         /*reserved*/
    boolean   reserved;                         /*reserved*/
                                                /*memory sizes are in bytes*/
    unsigned longint;                           /*preferred memory size*/
    unsigned longint;                           /*minimum memory size*/
};
```

标志域中的非保留位部分意义如下:

标志说明

acceptSuspendResumeEvents

设定后，就表明您的应用程序可以处理挂起和恢复执行事件(在把这些事件送到后台之前或带到前台之时，操作系统需将这些事件发送到您的应用程序中)。

附　注

如果您设定了acceptSuspendResumeEvents标志，您也应该设定doesActivateOnFGSwitch 标志。

canBackground　设定后，表示您的应用程序在后台时要接收空事件处理时间。如果您的应用程序在后台时无事可做，您不应该设定该标志。

doesActivateOnFGSwitch

设定后，表示您的应用程序在响应一个挂起或恢复执行事件时，需要负责激活或停用所有窗口。如果设定了acceptSuspendResnme-Events标志而没有设定doesActivateOnFGSwitch标志，而且您的应用程序处于挂起状态，那么您的应用程序在该挂起事件之后将接收一个激活事件。但如果您设定了 doesActivateOnFGSwitch 标志，那么您的应用程序将拒绝接收与操作系统事件相关联的激活事件，并且您必须在您的应用程序接收挂起或恢复执行事件时照顾到相应的激活或停用事件。这意味着如果应用程序的一窗口处于最前面，您应该象接收到一个停用事件那样对待一个挂起事件(假定DoesActivateOnFGSwitch和acceptSuspendResameEvent 标志均被设定)。例如，如果您的应用程序切换到了后台，您应该隐藏滚动杆，隐藏闪烁的插入光标，并取消所有选定文字的醒目显示。如果您不设定该标志，则进程管理器将产生一个窗口移位以强制激活和失活事件发生。

onlyBackground　设定后，表示您的应用程序仅跑在后台。这通常是因为它没有一个用户界面而不能在前台运行。

getFrontClicks　设定后，表示您的应用程序准备接收按下鼠标和释放鼠标事件，当用户在您的应用程序的最前面窗口点按时，这些事件会将您的应用程序带到前台。一般情况下，用户只想把您的应用程序带到前台，所以不希望接收此

类鼠标事件　这些鼠标事件有可能移动插入点或立即开始绘图，这由应用程序来决定)。Finder 是一个应用程序，它设定了 getFrontClicks 标志。

当用户在您应用程序的前面窗口点按并导致发生主切换时，您的应用程序会收到一个恢复执行事件。您的应用程序在响应恢复执行事件时应激活它的前面窗口。在这种情况下，如果应用程序的 getFrontClicks 标志没有设定，您的应用程序将拒绝接收导致主切换的相关鼠标事件。如果设定了应用程序的 getFrontClicks 标志，您的应用程序会接收到相关的鼠标事件。

当用户点按您应用程序中的一个非前台窗口时，您的应用程序总是接收到相关的鼠标事件并导致发生前台切换。

当您的应用程序在 System 7 中接收到一个鼠标按下事件时，它可以通过核查事件记录的 modifiers 域的 0 位来确定鼠标移动事件是否导致了一个前台切换。如果您的应用程序设定了 getFrontClicks 标志，此信息就会变得特别有用。例如，您的应用程序可以通过核查位 0 来决定是否处理该鼠标按下事件　很可能取决于被点按项目在前台切换前是否可视)。

acceptAppDiedEvents

设定后，表示每当您的应用程序终止或崩溃而启用一个应用程序时，您的应用程序都将得到通知。如果可以使用进程管理器，您的应用程序将把该事件当作一个苹果事件来接收，即应用程序死亡事件。请参阅《Inside Macintosh 进程》(*Inside Macintosh: Processes*) 一书中的“进程管理器”一章关于启用应用程序和接收应用程序死亡事件的进一步介绍。

附　注

MultiFinder 的一些早期版本不发送应用程序死亡事件。如果您的应用程序在 System 6 中运行，就未必能接收到应用程序死亡事件。这些事件主要供调试程序使用。

is32BitCompatible

设定后，表示您的应用程序可以在 32- 位存储器管理器中运行。除非您已在一个 32 位系统上　如以 32 位方式或在 A/UX 下运行 System 7 的一台 Macintosh II ci 电脑）充分地测试了您的应用程序，您不应该设定该标志。

isHighLevelEventAware

设定后，表示您的应用程序可以发送和接收高级事件。如果该标志没有设定，则当您调用 WaitNextEvent 时，事件管理器将拒绝为应用程序提供高级事件。没有屏蔽特殊类型的高级事件的方法；　如果设

定了此标志，您的应用程序就接收发送给应用程序所有类型的高级事件。

如果您设定了 isHighLevelEventAware 标志，您的应用程序必须支持4种所需的苹果事件。请参阅《Inside Macintosh应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中关于怎样响应4种所需的苹果事件的说明。

`localAndRemoteHLEvents`

设定后，表示您的应用程序对于在网络上的其它电脑中运行的应用程序（除在本地电脑中运行的应用程序外）来说，是可视的。如果该标志没有设定，您的应用程序将拒绝接收跨网络的高级事件。

`isStationeryAware`

设定后，表示您的应用程序可以识别固定样板文稿。如果没有设定该标志而用户打开一个固定样板文稿时，Finder将拷贝该文稿并提示用户输入拷贝文稿的名称。若想了解您的应用程序怎样才能使用固定样板文稿，请参阅本书的“Finder 界面”一章。

`useTextEditServices`

设定后，表示您的应用程序可以使用 TextEdit 提供的直接插入文字服务。请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书中关于 TextEdit 直接输入函数的介绍。

您指定的应用程序的优选和最小存储器大小取决于应用程序的特殊存储器要求。应用程序的存储器要求又取决于应用程序的静态堆、动态堆、A5 区域和堆栈的大小。（请参阅《Inside Macintosh: Memory》一书中“存储器管理介绍”部分，以便了解程序存储器分区中的这些区域的详细介绍。）

静态堆大小包括那些在应用程序执行期间总是出现的对象—例如 代码程序段，窗口记录的工具箱数据结构，等等。

动态堆要求取决于每个文稿产生的对象数目（其大小随文稿自身变化而成比例地变化），还取决于特定命令或函数所需的对象数目。

A5 区域的大小取决于全局数据的总量和应用程序包含的程序段间的转移次数。

最后，堆栈包含变量，返回地址，和临时信息。每台电脑的应用程序堆栈大小各不相同，因而您应根据 Macintosh Plus 上所需的堆栈大小（8KB）来设立您的堆栈大小基值。进程管理器将自动地调整您申请的存储器总量以补偿在不同机器上的不同堆栈大小。例如，如果您申请 512KB，则在具有较大系统设定堆栈大小的机器上将被分配更大的堆栈空间（约 16KB）。

事件管理器总结

Pascal 小结

常数

```
CONST     {event codes}
          nullEvent          = 0;         {no other pending events}
          mouseDown          = 1;         {mouse button pressed}
          mouseUp            = 2;         {mouse button released}
          keyDown            = 3;         {key pressed}
          keyUp              = 4;         {key released}
          autoKey            = 5;         {key repeatedly held down}
          updateEvt          = 6;         {window needs updating}
          diskEvt            = 7;         {disk inserted}
          activateEvt        = 8;         {activate/deactivate window}
          osEvt              = 15;        {operating-system events }
                                          { (suspend, resume, mouse-moved)}
          kHighLevelEvent    = 23;        {high-level events }
                                          { (includes Apple events)}

          {event masks}
          everyEvent         = -1;        {every event}
          mDownMask          =  2;        {mouse-down event      (bit 1)}
          mUpMask            =  4;        {mouse-up event        (bit 2)}
          keyDownMask        =  8;        {key-down event        (bit 3)}
          keyUpMask          =  16;       {key-up event          (bit 4)}
          autoKeyMask        =  32;       {auto-key event        (bit 5)}
          updateMask         =  64;       {update event          (bit 6)}
          diskMask           =  128;      {disk-inserted event   (bit 7)}
          activMask          =  256;      {activate event        (bit 8)}
          highLevelEventMask=  1024;      {high-level event      (bit 10)}
          osMask             = -32768;    {operating-system event (bit 15)}

          {message codes for operating-system events}
          suspendResumeMessage   = $01;        {suspend or resume event}
          mouseMovedMessage      = $FA;        {mouse-moved event}
          osEvtMessageMask       = $FF000000;  {can use to extract msg code}
```

```
{flags for suspend and resume events}
resumeFlag              = 1;      {resume event}
convertClipboardFlag    = 2;      {Clipboard conversion }
                                  { required}
{event message masks for keyboard events}
charCodeMask  = $000000FF;        {use to get character code}
keyCodeMask   = $0000FF00;        {use to get key code}
adbAddrMask   = $00FF0000;        {ADB address for ADB keyboard}

{constants corresponding to bits in the modifiers field of event}
activeFlag    = 1;           {bit 0 of low byte--valid only for }
                             { activate and mouse-moved events}
btnState      = 128;         {bit 7 of low byte is mouse button state}
cmdKey        = 256;         {bit 0 of high byte}
shiftKey      = 512;         {bit 1 of high byte}
alphaLock     = 1024;        {bit 2 of high byte}
optionKey     = 2048;        {bit 3 of high byte}
controlKey    = 4096;        {bit 4 of high byte}

{high-level event posting options}
nAttnMsg                = $00000001;  {give this message priority}
priorityMask            = $000000FF;  {mask for priority options}
nReturnReceipt          = $00000200;  {return receipt requested}
systemOptionsMask       = $00000F00;
receiverIDisTargetID    = $00005000;  {ID is port name & location}
receiverIDisSessionID   = $00006000;  {ID is PPC session ref number}
receiverIDisSignature   = $00007000;  {ID is creator signature}
receiverIDisPSN         = $00008000;  {ID is process serial number}
receiverIDMask          = $0000F000;

{class and ID values for return receipt}
HighLevelEventMsgClass = 'jaym';     {event class of return receipt}
rtrnReceiptMsgID        = 'rtrn';     {event ID of return receipt}

{modifiers values in return receipt}
msgWasNotAccepted           = 0;        {recipient did not accept }
                                        { the message}
msgWasFullyAccepted         = 1;        {recipient accepted the}
                                        { entire message}
msgWasPartiallyAccepted     = 2;        {recipient did not accept }
                                        { the entire message}
```

数据类型

```
TYPE
    EventRecord =
    RECORD
        what:          Integer;           {event code}
        message:       LongInt;           {event message}
        when:          LongInt;           {ticks since startup}
        where:         Point;             {mouse location}
        modifiers:     Integer;           {modifier flags}
    END;

    KeyMap = PACKED ARRAY[0..127] OF Boolean;{records state of keyboard}

    TargetID =
    RECORD
        sessionID:     LongInt;           {session reference number (not }
                                          { used if posting an event)}
        name:          PPCPortRec;        {port name}
        location:      LocationNameRec;   {location name}
        recvrName:     PPCPortRec;        {reserved}
    END;

    TargetIDPtr        = ^TargetID;       {pointer to a target ID record}
    TargetIDHdl        = ^TargetIDPtr;    {handle to a target ID record}
    HighLevelEventMsg =
    RECORD
        HighLevelEventMsgHeaderLength:  Integer;       {reserved}
        version:                        Integer;       {reserved}
        reserved1:                      LongInt;       {reserved}
        theMsgEvent:                    EventRecord;   {event record}
        userRefCon:                     LongInt;       {reference constant}
        postingOptions:                 LongInt;       {reserved}
        msgLength:                      LongInt;       {reserved}
    END;

    HighLevelEventMsgPtr       = ^HighLevelEventMsg;
    HighLevelEventMsgHdl       = ^HighLevelEventMsgPtr;

    GetSpecificFilterProcPtr   = ProcPtr;
```

```
EvQEl =                              {event queue entry}
RECORD
     qLink:         QElemPtr;        {next queue entry}
     qType:         Integer;         {queue type (ORD(evType))}
     evtQWhat:      Integer;         {event code}
     evtQMessage:   LongInt;         {event message}
     evtQWhen:      LongInt;         {ticks since startup}
     evtQWhere:     Point;           {mouse location}
     evtQModifiers:Integer;          {modifier flags}
END;

EvQElPtr = ^EvQEl;
```

事件管理器例程

接收事件

```
FUNCTION WaitNextEvent      (eventMask: Integer; VAR theEvent: EventRecord;
                            sleep: LongInt; mouseRgn: RgnHandle): Boolean;
FUNCTION EventAvail         (eventMask: Integer; VAR theEvent: EventRecord)
                            : Boolean;
FUNCTION GetNextEvent       (eventMask: Integer; VAR theEvent: EventRecord)
                            : Boolean;
FUNCTION AcceptHighLevelEvent
                            (VAR sender: TargetID; VAR msgRefcon: LongInt;
                            msgBuff: Ptr; VAR msgLen: LongInt): OSErr;
FUNCTION GetSpecificHighLevelEvent
                            (aFilter: GetSpecificFilterProcPtr;
                            yourDataPtr: UNIV Ptr; VAR err: OSErr)
                            : Boolean;
PROCEDURE FlushEvents       (whichMask: Integer; stopMask: Integer);
PROCEDURE SystemClick       (theEvent: EventRecord; theWindow: WindowPtr);
PROCEDURE SystemTask;
FUNCTION SystemEvent        (theEvent: EventRecord): Boolean;
FUNCTION GetOSEvent         (mask: Integer; VAR theEvent: EventRecord)
                            : Boolean;
FUNCTION OSEventAvail       (mask: Integer; VAR theEvent: EventRecord)
                            : Boolean;
PROCEDURE SetEventMask      (theMask: Integer);
FUNCTION GetEvQHdr          : QHdrPtr;
```

发送事件

```
FUNCTION PostHighLevelEvent(theEvent: EventRecord; receiverID: Ptr;
                            msgRefcon: LongInt; msgBuff: Ptr;
                            msgLen: LongInt; postingOptions: LongInt)
                            : OSErr;
FUNCTION PPostEvent         (eventCode: Integer; eventMsg: LongInt;
                            VAR qEl: EvQElPtr): OSErr;
FUNCTION PostEvent          (eventNum: Integer; eventMsg: LongInt): OSErr;
```

转换进程序号和端口名称

```
FUNCTION GetProcessSerialNumberFromPortName
                            (portName: PPCPortRec;
                            VAR PSN: ProcessSerialNumber): OSErr;
FUNCTION GetPortNameFromProcessSerialNumber
                            (VAR portName: PPCPortRec;
                            PSN: ProcessSerialNumber): OSErr;
```

读鼠标

```
PROCEDURE GetMouse          (VAR mouseLoc: Point);
FUNCTION Button             : Boolean;
FUNCTION StillDown          : Boolean;
FUNCTION WaitMouseUp        : Boolean;
```

读键盘

```
PROCEDURE GetKeys           (VAR theKeys: KeyMap);
{the KeyTranslate function is also available as the KeyTrans function}
FUNCTION KeyTranslate       (transData: Ptr; keycode: Integer;
                            VAR state: LongInt): LongInt;
```

获取时间信息

```
FUNCTION TickCount          : LongInt;
FUNCTION GetDblTime         : LongInt;
FUNCTION GetCaretTime       : LongInt;
```

应用程序定义的例程

搜索高级事件队列的过滤函数

```
FUNCTION MyFilter           (yourDataPtr: Ptr;
                            msgBuff: HighLevelEventMsgPtr;
                            sender: TargetID): Boolean;
```

C 小结

常数

```
enum {
    /*event codes*/
    nullEvent              = 0,     /*no other pending events*/
    mouseDown              = 1,     /*mouse button pressed*/
    mouseUp                = 2,     /*mouse button released*/
    keyDown                = 3,     /*key pressed*/
    keyUp                  = 4,     /*key released*/
    autoKey                = 5,     /*key repeatedly held down*/
    updateEvt              = 6,     /*window needs updating*/
    diskEvt                = 7,     /*disk inserted*/
    activateEvt            = 8,     /*activate/deactivate window*/
    osEvt                  = 15,    /*operating-system events */
                                    /* (suspend, resume, mouse-moved)*/

    /*event masks*/
    mDownMask              =  2,    /*mouse-down          (bit 1)*/
    mUpMask                =  4,    /*mouse-up            (bit 2)*/
    keyDownMask            =  8,    /*key-down            (bit 3)*/
    keyUpMask              =  16,   /*key-up              (bit 4)*/
    autoKeyMask            =  32,   /*auto-key            (bit 5)*/
    updateMask             =  64,   /*update              (bit 6)*/
    diskMask               =  128,  /*disk-inserted       (bit 7)*/
    activMask              =  256,  /*activate            (bit 8)*/
    highLevelEventMask     =  1024, /*high-level          (bit 10)*/
    osMask                 = -32768 /*operating-system    (bit 15)*/
};

enum {
    everyEvent                  = -1,     /*every event*/

    /*event message masks for keyboard events*/
    charCodeMask        = 0x000000FF,     /*use to get character code*/
    keyCodeMask         = 0x0000FF00,     /*use to get key code*/
    adbAddrMask         = 0x00FF0000,     /*ADB address if ADB keyboard*/
    osEvtMessageMask    = 0xFF000000,     /*can use to extract msg code*/
```

```
        /*message codes for operating-system events*/
        mouseMovedMessage      = 0xFA,      /*mouse-moved event*/
        suspendResumeMessage   = 0x01,      /*suspend or resume event*/
        /*flags for suspend and resume events*/
        resumeFlag             = 1,         /*resume event*/
        convertClipboardFlag   = 2,         /*Clipboard conversion */
                                            /* required*/

        /*constants corresponding to bits in the modifiers field of event*/
        activeFlag     = 1,      /*bit 0 of low byte--valid only for */
                                 /* activate and mouse-moved events*/
        btnState       = 128,    /*bit 7 of low byte is mouse button state*/
        cmdKey         = 256,    /*bit 0 of high byte*/
        shiftKey       = 512,    /*bit 1 of high byte*/
        alphaLock      = 1024,   /*bit 2 of high byte*/
        optionKey      = 2048,   /*bit 3 of high byte*/
        controlKey     = 4096    /*bit 4 of high byte*/
};
enum {
        kHighLevelEvent    = 23,          /*event code for high-level events */
                                          /* (includes Apple events)*/
        /*high-level event posting options*/
        receiverIDMask      = 0x0000F000, /*mask for receiver ID bits*/
        receiverIDisPSN     = 0x00008000, /*ID is proc serial number*/
        receiverIDisSignature= 0x00007000, /*ID is creator signature*/
        receiverIDisSessionID = 0x00006000, /*ID is session ref number*/
        receiverIDisTargetID= 0x00005000, /*ID is port name & location*/

        systemOptionsMask = 0x00000F00,
        nReturnReceipt      = 0x00000200, /*return receipt requested*/
        priorityMask        = 0x000000FF, /*mask for priority options*/
        nAttnMsg            = 0x00000001, /*give this message priority*/

        /*class and ID values for return receipt*/
        #define HighLevelEventMsgClass 'jaym'
        #define rtrnReceiptMsgID 'rtrn'

        /*modifiers values in return receipt*/
        msgWasPartiallyAccepted     = 2,
        msgWasFullyAccepted         = 1,
        msgWasNotAccepted           = 0
};
```

数据类型

```
struct   EventRecord {
    short    what;                    /*event code*/
    long     message;                 /*event message*/
    long     when;                    /*ticks since startup*/
    Point    where;                   /*mouse location*/
    short    modifiers;               /*modifier flags*/
    };
typedef struct EventRecord EventRecord;

typedef long KeyMap[4];               /*records state of keyboard*/

struct   TargetID {
    long             sessionID;       /*session reference number (not */
                                      /* used if posting an event)*/
    PPCPortRec       name;            /*port name*/
    LocationNameRec  location;        /*location name*/
    PPCPortRec       recvrName;       /*reserved*/
    };

typedef struct    TargetID TargetID;
typedef TargetID *TargetIDPtr, **TargetIDHdl;

struct   HighLevelEventMsg {
    unsigned short   HighLevelEventMsgHeaderLength;     /*reserved*/
    unsigned short   version;                           /*reserved*/
    unsigned long    reserved1;                         /*reserved*/
    EventRecord      theMsgEvent;                       /*event record*/
    unsigned long    userRefCon;                        /*ref constant*/
    unsigned long    postingOptions;                    /*reserved*/
    unsigned long    msgLength;                         /*reserved*/
    };

typedef struct    HighLevelEventMsg HighLevelEventMsg;
typedef HighLevelEventMsg *HighLevelEventMsgPtr, **HighLevelEventMsgHdl;

    struct EvQEl {    /*event queue entry*/
        QElemPtr     qLink;                /*next queue entry*/
        short        qType;                /*queue type (evType)*/
        short        evtQWhat;             /*event code*/
        long         evtQMessage;          /*event message*/
        long         evtQWhen;             /*ticks since startup*/
```

```
        Point         evtQWhere;              /*mouse location*/
        short         evtQModifiers;          /*modifier flags*/
    };
    typedef struct EvQEl EvQEl;
    typedef EvQEl *EvQElPtr;

typedef pascal Boolean (*GetSpecificFilterProcPtr)
                        (void *yourDataPtr,
                        HighLevelEventMsgPtr msgBuff,
                        const TargetID *sender);
```

事件管理器例程

接收事件

```
pascal Boolean WaitNextEvent    (short eventMask, EventRecord *theEvent,
                                unsigned long sleep, RgnHandle mouseRgn);
pascal Boolean EventAvail       (short eventMask, EventRecord *theEvent);
pascal Boolean GetNextEvent     (short eventMask, EventRecord *theEvent);
pascal OSErr AcceptHighLevelEvent
                                (TargetID *sender, unsigned long *msgRefcon,
                                Ptr msgBuff, unsigned long *msgLen);
pascal Boolean GetSpecificHighLevelEvent
                                (GetSpecificFilterProcPtr aFilter,
                                void *yourDataPtr, OSErr *err);
pascal void FlushEvents         (short whichMask, short stopMask);
pascal void SystemClick         (const EventRecord *theEvent,
                                WindowPtr theWindow);
pascal void SystemTask          (void);
pascal Boolean SystemEvent      (const EventRecord *theEvent);
pascal Boolean GetOSEvent       (short mask, EventRecord *theEvent);
pascal Boolean OSEventAvail     (short mask, EventRecord *theEvent);
pascal void SetEventMask        (short theMask);
#define GetEvQHdr()             ((QHdrPtr) 0x014A)
```

发送事件

```
pascal OSErr PostHighLevelEvent
                        (const EventRecord *theEvent,
                        unsigned long receiverID,
                        unsigned long msgRefcon, Ptr msgBuff,
                        unsigned long msgLen,
                        unsigned long postingOptions);
```

```
pascal OSErr PPostEvent     (short eventCode, long eventMsg, EvQElPtr *qEl)
pascal OSErr PostEvent      (short eventNum, long eventMsg);
```

转换进程序号和端口名称

```
pascal OSErr GetPortNameFromProcessSerialNumber
                       (PPCPortPtr portName,
                       const ProcessSerialNumberPtr pPSN);
pascal OSErr GetProcessSerialNumberFromPortName
                       (const PPCPortPtr portName,
                       ProcessSerialNumberPtr pPSN);
```

读鼠标

```
pascal void GetMouse        (Point *mouseLoc);
pascal Boolean Button       (void);
pascal Boolean StillDown    (void);
pascal Boolean WaitMouseUp  (void);
```

读键盘

```
pascal void GetKeys         (KeyMap theKeys);
{the KeyTranslate function is also available as the KeyTrans function}
pascal long KeyTranslate    (const void *transData, short keycode,
                            long *state);
```

获取时间信息

```
pascal unsigned long TickCount
                            (void);
#define GetDblTime()        (* (unsigned long*) 0x02F0)
#define GetCaretTime()      (* (unsigned long*) 0x02F4)
```

应用程序定义的例程

用于搜索高级事件队列的过滤函数

```
pascal Boolean MyFilter     (void *yourDataPtr,
                            HighLevelEventMsgPtr msgBuff,
                            const TargetID *sender);
```

汇编语言小结

数据结构

事件数据结构

0	what	字	事件代码
2	message	长型	事件信息
6	when	长型	自启动以来的时间（以 Tick 为单位）
10	where	长型	鼠标位置信息
14	modifiers	字	修正键标志位

目标标识符 数据结构

0	sessionID	长型	对话项目参考编号（发送事件时不用）
4	name	68 字节	端口名称（在 PPCPortRec 数据结构中指定）
72	location	34 字节	位置名称（在 LocationNameRec 中指定）
106	recvrName	68 字节	保留

高级事件信息数据结构

0	HighLevelEventMsgHeaderLength	字	保留
2	version	字	保留
4	reserved1	长型	保留
8	theMsgEvent	16 字节	事件记录
22	userRefCon	长型	参考长数
26	postingOptions	长型	保留
30	msgLength	长型	保留

事件队列首部数据结构

0	qLink	长型	下一个队列入口地址
4	qType	字	队列类型
6	evtQWhat	字	事件代码
8	evtQMessage	长型	事件信息
12	evtQWhen	长型	自启动以来的时间（以 Tick 为单位）
16	evtQWhere	长型	鼠标位置信息
20	evtQModifiers	字	修正键标志位

陷阱宏

陷阱宏要求例程选择程序

_OSDispatch

选择程序	例程
$0033	AcceptHighLevelEvent
$0034	PostHighLevelEvent

选择程序	例程
$0035	GetProcessSerialNumberFromPortName
$0045	GetSpecificHighLevelEvent
$0046	GetPortNameFromProcessSerialNumber

陷阱宏要求寄存器设置

陷阱宏名称	寄存器输入		寄存器退出	
_FlushEvents	D0：	事件类型屏蔽参数（低位字）停止屏蔽参数（高位字）	D0：	如果从队列中删除所有的事件，或停止搜索事件的事件代码（低位字），则为0。
_GetOSEvent	A0：	事件记录的地址：		
	D0：	事件类型屏蔽参数（低位字）	D0：	如果GetOSEvent返回非空事件为0；如果返回空事件（低位字节）则为-1。
_OSEventAvail	A0：	事件记录的地址：		
	D0：	事件类型屏蔽参数（低位字）	D0：	如果OSEventAvail返回非空事件为0；如果返回空事件（低位字节）则为-1。
_PostEvent	A0：	事件记录的地址：		
	D0：	事件消息　长型字）	D0：	结果代码（低位字）

全局变量

caretTime	可编辑文字中插入光标闪烁间之建议Tick差.
DoubleTime	释放鼠标事件和鼠标按下事件之间的建议最大Tick 差，您的应用程序可以将此Tick差内的两个鼠标事件视为双击事件。
EventQueue	事件队列头。
KeyRepThresh	自动键速率值　在事件管理器生成一个后续自动键事件之前所必需的时间Tick 数）
KeyThresh	自动键阈值　在事件管理器生成一个自动键事件之前所必需的时间Tick 数)。
ScrDmpEnable	一个字节。如果设定为0，则会令事件管理器对Command-Shift一数字3至9键组合的处理失效。
SEvtEnb	一个字节。如果设定为0，则SystemEvent函数总是返回FALSE　假)。
SysEvtMask	当前应用程序的系统事件类型屏蔽参数。
Ticks	表示自系统最后一次启动以来的当前Tick 数的长整数。

结果代码

noErr	0	没有错误
procNotFound	–600	没有带指定进程序号的可选入进程缓冲区太小
bufferIsSmall	–607	没有未完成的高级事件
noOutstandingHle	–608	连接无效
noUserInteractionAllowed	–610	不能直接与用户对话
noPortErr	–903	无效端口名称

第三章　菜单管理器

菜单介绍	3–5
菜单和菜单栏定义例程	3–9
菜单栏	3–9
菜单	3–11
菜单命令项目	3–12
菜单命令项目分组	3–14
菜单命令的等效命令组合键	3–16
菜单管理器自动添加菜单	3–19
苹果菜单	3–20
文件菜单	3–22
编辑菜单	3–24
字体菜单	3–26
大小菜单	3–27
帮助菜单	3–29
键盘菜单	3–32
应用程序菜单	3–33
弹出式菜单	3–33
嵌套式菜单	3–38
关于菜单管理器	3–39
菜单管理器如何保留有关菜单的信息	3–40
菜单管理器如何保留有关应用程序菜单栏的信息	3–40
使用菜单管理器	3–41
创建菜单	3–42
创建菜单资源	3–43
创建菜单栏资源	3–49
设置应用程序的菜单栏	3–50
创建嵌套式菜单	3–53
创建弹出式菜单	3–56

第三章

菜单管理器

更改菜单中的项目外观 3-57
启用和停用菜单命令项目 3-58
更改项目文字 3-59
更改菜单命令项目的字形 3-60
更改菜单命令项目的标记 3-61
更改菜单命令项目的图标或语系输入代码 3-62
添加菜单命令项目 3-64
在帮助菜单中添加项目 3-67
在苹果菜单中添加项目 3-68
添加字体菜单 3-69
处理用户选择的菜单命令 3-70
处理菜单栏中的鼠标按下事件 3-72
调整应用程序菜单 3-73
确定用户是否选择了等效命令组合键 3-77
响应用户选择的菜单命令项目 3-78
处理苹果菜单 3-80
处理帮助菜单 3-81
处理大小菜单 3-82
从对话框中存取菜单 3-84
编写您自己的菜单定义过程 3-87
计算菜单的尺寸 3-89
绘制菜单中的菜单命令项目 3-90
确定光标是否位于启用菜单命令项目 3-92
菜单管理器参考资料 3-95
数据结构 3-95
菜单记录 3-95
菜单列表 3-97
菜单色彩信息表记录 3-98
菜单管理器例程 3-102
初始化菜单管理器 3-103
创建菜单 3-105
向或从当前菜单列表中添加或删除菜单 3-108
从'MBAR'资源中获取菜单栏说明信息 3-110
获取和设定菜单栏 3-112
绘制菜单栏 3-113
响应用户选择的菜单命令 3-114
为菜单记录获取一个间接指针 3-122
添加和删除菜单命令项目 3-124
获取和设定菜单命令项目的外观 3-130
排列菜单 3-140
计算菜单中的项目数 3-140
醒目显示菜单栏 3-141
重新计算菜单尺寸 3-142
管理菜单色彩信息表中的输入 3-143
应用程序定义的例程 3-148
菜单定义过程 3-148

第三章

菜单管理器

资源 3-151
菜单资源 3-151
菜单栏资源 3-155
菜单色彩信息表资源 3-155
菜单定义过程资源 3-157
菜单管理器小结 3-158
Pascal 小结 3-158
常数 3-158
数据类型 3-158
菜单管理器例程 3-159
应用程序定义的例程 3-162
C 小结 3-162
常数 3-162
数据类型 3-163
菜单管理器例程 3-164
应用程序定义的例程 3-166
汇编语言小结 3-167
数据结构 3-167
全局变量 3-167
结果代码 3-167

第三章

菜单管理器

您可以使用菜单管理器创建和管理应用程序中的菜单。菜单允许用户显示或选择应用程序提供的选择列表和命令列表。

所有 Macintosh 应用程序均应提供这三个标准菜单：苹果菜单、文件菜单和编辑菜单。如果您的应用程序菜单中拥有一个苹果菜单，菜单管理器就会在您的应用程序菜单栏中自动地添加帮助和应用程序菜单；如果安装了多种键盘布局或输入方法，还可以添加键盘输入菜单。

菜单一般是作为资源储存的。本章介绍了与菜单相关的资源。请参阅本书“Macintosh 工具箱介绍”一章中关于资源的一般信息，并请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书的“资源管理器”一章中关于资源管理器例程的信息，《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书中的一些附加菜单实例，这些菜单实例综合了多种用户界面设计准则。《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书中有一些关于您的应用程序在国际市场定位的详细信息。

您可以为您的应用程序菜单选择提供旁述的帮助。请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书“帮助管理器”一章中关于为您的应用程序菜单提供旁述的附加说明信息。

如果某一条菜单命令在执行之前需要征询用户一些附加信息，通常就需要为用户提供一个对话框。请参阅本书后面的“对话管理器”一章中关于在您的应用程序中创建对话框的介绍。

如果希望了解有关处理事件的附加信息，请参阅本书前面的“事件管理器”一章。

- 您的应用程序可以使用的各种菜单类型
- 标准菜单
- 如何将菜单作为资源来储存
- 如何创建菜单
- 如何创建菜单栏
- 如何更改菜单命令项目的特性
- 如何为菜单添加命令项目

菜单介绍

菜单是一种用户界面元素，您可以在您的应用程序中使用该元素，以便用户能够在应用程序提供的选择列表和命令列表中显示或选择命令项目。菜单可以几种不同的形式出现：下拉式菜单、嵌套式菜单和弹出式菜单。

下拉式菜单由菜单栏中的菜单标题（一个单词或一个图标）来标识。您的应用程序可以在菜单栏中使用下拉式菜单，以便让用户能够为选定对象选择一条命令或执行一个动作。弹出式菜单不出现在菜单栏中，但当光标处于某特殊位置、用户按下鼠标按钮时，该菜单便会出现在屏幕上的某个位置上。多数情况是访问对话框中的弹出式菜单。您的应用程序可以利用弹出式菜单让用户从选择列表中的多个选项中选择一个项目或设定一个特定值。子菜单是指附连在其它菜单上的菜单。附连有子菜单的菜单称为嵌套式菜单。

图 3 － 1 是下拉式菜单、子菜单和弹出式菜单的实例。

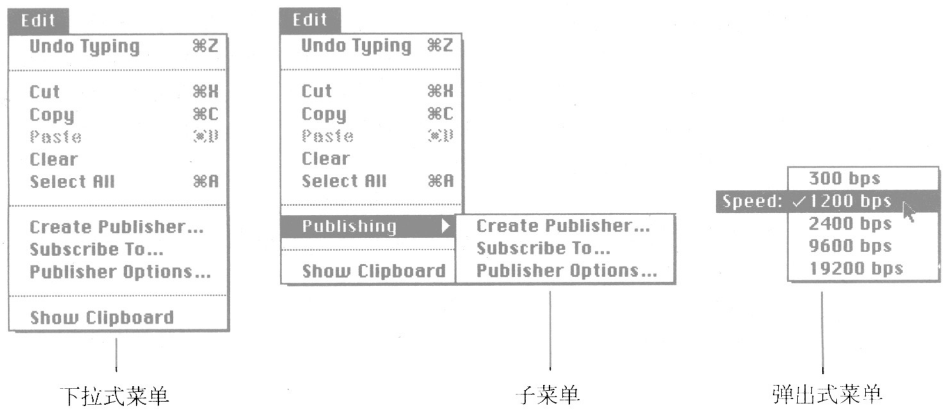


图 3-1　下拉式菜单、子菜单和弹出式菜单

标准菜单栏一直延伸到屏幕的顶部并包括每个下拉式菜单的标题。您的应用程序菜单栏至少应提供苹果菜单、文件菜单和编辑菜单等标准菜单。当把苹果菜单插入应用程序的菜单栏时，菜单管理器会自动将帮助菜单和应用程序菜单添加到您的应用程序的菜单栏中。另外，如果安装了多种语系系统或在'itlc'资源中设定了某个特定标志位，您的应用程序还可以添加键盘菜单。在您的应用程序中应该尽可能地多设置一些菜单，当然要适度。您应该只创建您的应用程序所必须的那些基本命令项目的资源。

如果您的应用程序使用菜单栏，就应让用户总能看到和使用它。如果您不希望总是显示菜单栏（例如，如果您的应用程序允许用户显示屏幕表示），您应该为用户提供能够显示整个屏幕表示的选项，而无需总显示菜单栏。但是，您必须提供一种有效方法，如等效命令组合键，以便用户存取菜单栏或重新显示菜单栏。

在您的应用程序中使用菜单，可以使用户尝试各选项的作用。一般可以通过在菜单中提供旁述，让用户进一步了解选择某个菜单选项可能引起的动作和可能导致的结果，而不必真正选择菜单命令来弄清楚可能造成的后果。

图3－2是一个 SurfWriter 应用程序菜单栏，在该菜单栏中显示的是编辑菜单。此应用程序支持标准的苹果菜单、文件菜单和编辑菜单；帮助菜单和应用程序菜单；另外还支持两个应用程序专用菜单。

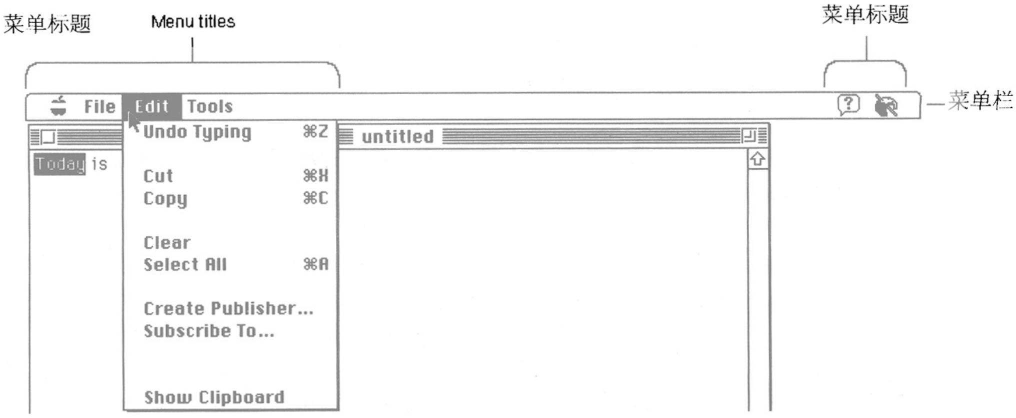


图3-2　显示编辑菜单的 surfWriter 应用程序菜单栏

每个菜单都有一个菜单标题和一个或多个与之相关的菜单命令项目。您应该为每个菜单命名，以便使得菜单标题能够说明该菜单可以执行的动作。例如，典型的应用程序编辑菜单中含有允许用户编辑文稿内容的命令。

您的应用程序可以停用任何菜单。菜单管理器会将菜单标题的色彩变淡以表示该菜单被停用。（在图3－2中，彩色菜单被停用。）菜单管理器可以淡化停用菜单所有菜单命令项目的彩色。用户依然可以下拉和查看停用菜单中的项目，但不能使其中的停用项目动作。

您的应用程序也可以停用单个菜单命令项目。菜单管理器可以淡化停用项目的外观，而且当光标停留在该项目上时也不会醒目显示它。当光标位于停用菜单命令项目上时，如果用户释放鼠标按钮，菜单管理器将报告该用户未选择菜单命令。（但您可以利用 MenuChoice 函数来确定是否发生此事件。）

在图3－2中，贴命令被停用。如果剪贴板是空的，SurfWriter应用程序就停用贴命令。如果当前选定项中没有包含发行或引用程序，那么SurfWriter也会停用版本选项命令。正如《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*)一书中“帮助管理器”一章解释的那样，您的应用程序应针对停用项目提供旁述，说明该项目正常的操作功能并解释为什么此时该项目不能使用。

附注

虽然 enabled 和 disabled 是您在资源文件中用来醒目显示或淡化菜单和菜单命令项目色彩的常数，但是您不应在旁述或用户说明文件中使用这些术语。您可以使用"菜单"（menu）、"菜单命令"（menu Command）和"菜单命令项目"（menu item）等术语来表示那些有效的命令项目，"不可用"（not available）和"暗淡的"（dimmed）等术语来区别那些无效项目。

当光标位于菜单有效命令项目时，菜单管理器醒目显示该项目。用户可以进行选择它们。

在首次定义和创建菜单时，您的应用程序可以指定有效或无效的菜单命令项目。创建菜单后，您也可以随时停用或启用菜单命令项目。每当应用程序允许用户选择与该项目有关的动作时，则应启用该菜单命令项目，而当用户不应选择某项目时，则应停用该项目。例如，如果用户选择文字，然后点开菜单钮，您就应该启用编辑菜单中的复制命令。如果用户没有选择任何东西，您就应该停用编辑菜单中的复制命令。

您的应用程序也可以指定菜单命令项目的其它特征，如在项目文字旁是否有标志符号（例如，"✓"标记）或者菜单命令项目是否有等效命令组合键（例如，Command － C 为复制命令的等效命令组合键）。第 3 － 12 页的"菜单命令项目"，将更为详尽地介绍了单个菜单命令项目特征。

用户通常是将光标移至菜单栏并当光标位于菜单标题上时按下鼠标按钮拉下该菜单，然后选择命令。当光标位于菜单栏上而且用户按下了鼠标按钮时，您的应用程序调用 Menu Select 函数。随着用户在菜单栏中拖拽光标，MenuSelect 函数应跟踪鼠标、及时显示或取消菜单，当用户将光标拖拽到菜单标题上时，应醒目显示菜单标题，当用户在一个菜单中拖拽光标时，则应显示与选定菜单相关的菜单命令项目、醒目显示有效菜单命令项目，并处理所有的用户活动直至用户释放鼠标按钮。

当光标位于某特定的有效菜单命令项目上时，用户释放鼠标按钮即可选择一个菜单命令项目。在用户选择了菜单命令项目后，菜单管理器会暂短闪烁所选的菜单命令项目（以确认选择事件），然后消去该菜单。

MenuSelect 函数可以返回确认信息以便您的应用程序确定哪个菜单命令项目已被选定。您的应用程序一般是通过执行相应的命令来进行响应。当您的应用程序完成了要求的动作后，就不应再醒目显示该菜单标题，这样可以让用户知道该动作已经完成。

用户可以随时将光标移出菜单（或菜单栏）；只要按下鼠标按钮，菜单管理器便会显示当前的任何可视菜单，（如果光标位于菜单之外，菜单管理器将会取消所有醒目显示的菜单命令项目。）如果用户在光标位于菜单之外时释放鼠标按钮，MenuSelect 函数则报告用户未选择菜单命令项目，然后菜单管理器就会消去当前的所有可视菜单。如果用户没有选择菜单命令项目，您的应用程序就不应执行任何动作。

菜单和菜单栏定义例程

菜单定义过程和菜单栏定义函数可以定义菜单的一般外观和行为。菜单管理器可以使用这些例程显示菜单和菜单栏，并执行其基本操作。

菜单定义过程可以执行一个菜单内所有菜单命令项目的绘制任务。如果您定义了一个菜单，您就可以指定其菜单定义过程。菜单管理器可以使用指定的菜单定义过程绘制菜单中的菜单命令项目、确定用户从菜单中选择了哪个命令项目、将滚动指示符作为项目插入菜单，并计算菜单的尺寸。

菜单栏定义函数可以绘制菜单栏。当用户在显示菜单中移动光标时，菜单栏定义函数还能够执行与菜单显示相关的绘制活动。如果没有作特别指定，菜单管理器则利用标准的菜单栏定义函数来管理您的应用程序菜单栏。菜单管理器可以使用标准的菜单栏定义函数来绘制菜单栏、清除菜单栏、确定光标是否在菜单栏上或在任何当前可视的菜单上、计算菜单标题的左边缘、醒目显示菜单标题、以反色显示整个菜单栏、抹掉菜单的背景色和绘制菜单框架（阴影）、以及存储或还原菜单。

苹果可以提供标准的菜单定义过程和标准的菜单栏定义函数。这些定义例程均被作为资源储存在系统文件中。标准的菜单定义过程是资源标识符为 0 的'MDEF'资源。标准的菜单栏定义函数是资源标识符为 0 的'MBDF'资源。

当您定义菜单和菜单栏时，应指定定义例程，因为菜单管理器在管理菜单和菜单栏时要使用这些定义例程。通常，软件开发商都希望在他们的应用程序中使用标准的定义例程。但是，如果标准的菜单定义过程不能提供您需要的特性（例如，如果您希望在菜单中包含更多的图形）。您也可以编写您自己的菜单定义过程。请参阅从第3－87页开始的“编写您自己的菜单定义过程”中的详细介绍。虽然菜单管理器允许您指定自己的菜单栏定义函数，但苹果公司建议您最好使用标准的菜单栏定义函数。

菜单栏

每个应用程序都有它们自己的菜单栏。应用程序的菜单栏仅限该应用程序使用。通常利用一个菜单栏（'MBAR'）资源（这一资源列出了菜单栏中出现的每个菜单的顺序和资源标识符）就可以定义您的应用程序的菜单栏。通过为菜单栏中出现的每个菜单提供一个菜单（'MENU'）资源，就可以定义菜单标题和菜单中菜单命令项目的个别特征。您可以这些资源定义为基础，利用菜单管理器例程来创建菜单和菜单栏。

您的应用程序可以更改菜单的有效状态、在菜单栏上添加或删除菜单、或者更改任何菜单命令项目的特征。每当您的应用程序更改菜单的有效状态或其中的菜单数目时，您的应用程序必须调用 DrawMenuBar 过程来更新菜单栏的外观。

菜单栏为白色(标准菜单栏定义函数定义的)，其高度足以显示用系统字体高度和系统字体大小显示的菜单标题，其黑色底边界为一个像素高。菜单栏与屏幕同宽，当用户将显示器设为启动屏幕时，菜单栏总是出现在显示器上。(用户可以使用显示器控制板来选择启动屏幕。)菜单栏显示在屏幕的顶部，在其前面除光标之外没有其它任何东西。图 3 - 3 是 SurfWriter 应用程序的菜单栏。



图 3-3　SurfWriter 应用程序的菜单栏

菜单栏可用来帮助指示前台应用程序。前台应用程序是指其菜单栏当前正在显示，而且其图标出现在应用程序菜单的菜单标题处。

菜单标题显示在菜单栏上。菜单标题是一个字符串（除“苹果”、“帮助”，和“应用程序菜单”之外，它们的标题中包含一个小图标）。菜单标题总是按系统字体和系统字体大小来显示（对于罗马语系，系统字体是 Chicago，系统字体大小是 12）。

您可以在菜单栏中插入任意数量的菜单标题；但是，10 个以下的标题较为适当。请记住 ，用户使用的显示器的大小不一定相同。设计您的菜单栏时，应让所有的标题都能在您的应用程序运行的最小的屏幕上显示出来。当设计适合您的菜单栏中的菜单数目时，您还要考虑到本地化的问题 - 经过转换后的菜单标题不一定都能在菜单栏上显示出来。例如，当英文文字转换成其它语言时，其尺寸大小常常增长 50%。

图 3 - 4 是 SurfWriter 应用程序的菜单栏，带有菜单标题，而且其标题已用另一种语系系统本地化了。

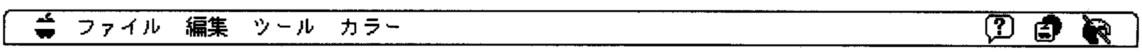


图 3-4　用另一种语系系统本地化的 SurfWriter 应用程序的菜单栏

菜单

菜单（标准菜单定义过程定义的）是一个菜单命令项目列表，菜单命令项目垂直排列在一个矩形框中。该矩形框加有底纹，能够以屏幕长度垂直延伸。如果菜单中的项目数量多于屏幕所能显示的数量，标准菜单定义过程会在屏幕上的最后一个项目上添加一个指向下方的三角形指示符，当用户将光标移过屏幕上当前显示的最后一个菜单命令项目时，余下的项目可以自动地滚动出来。当用户开始滚动菜单时，标准菜单定义过程则在屏幕的顶部项目上添加一个指向上方的三角形指示符，以指示用户可以将菜单卷回其原始位置。

每个菜单均可拥有与之相关的彩色信息。如果您没有在您的应用程序菜单色彩信息表中为您的菜单定义色彩，那么菜单管理器就会将系统设定色彩用作菜单和菜单栏的色彩。系统设定色彩是白底黑文。系统设定色彩一般都可以满足应用程序的需要。在3 － 98页上的“菜单色彩信息表记录”和3 － 155页上的“菜单彩色信息表资源”中有关于如何定义您的应用程序菜单彩色的说明。

您的应用程序菜单中可以包含任意数目的菜单命令项目。“菜单命令项目”（下一节）中介绍了您定义菜单命令项目时可以利用的视觉变化类型。

一般情况下，您会在一个'MBAR'资源中定义应用程序菜单栏中菜单的顺序和资源标识符。您应在'MBAR'资源中将苹果菜单定为菜单栏中的第一个菜单，其次应定义的两个菜单是文件和编辑菜单，然后再定义您的应用程序使用的其它菜单。您不必在'MBAR'资源中定义“键盘”、“帮助”或“应用程序”菜单；如果您的应用程序调用 GetNewMBar 函数而且您的菜单栏中包含苹果菜单，或者您的应用程序使用 InsertMenu 过程将苹果菜单插入当前菜单列表，那么菜单管理器会自动地将“键盘”、“帮助”或“应用程序”菜单添加到您的应用程序的菜单栏中。

在'MENU'资源中，您可以定义菜单标题和每个单独的菜单命令项目的特征。3 － 43页中的“创建菜单资源”详细介绍了'MENU'资源。

弹出式菜单不出现在菜单栏中，但可在屏幕的其它地方显示。您通常需要在对话框中使用弹出式菜单，因为这时您总是希望用户能够在一个较大的选择列表中进行选择。例如，您可以在用户方便时使用弹出式菜单来显示选项，而不是用多个选项按钮来显示选项。

嵌套式菜单是指带有子菜单的下拉式菜单或弹出式菜单。（但是，您应尽量避免在弹出式菜单中挂接子菜单，因为这样会使界面更为复杂而且对用户来说不够直观。）

在3－56页的“创建弹出式菜单”中有关于弹出式菜单的附加说明，在3－53页的“创建嵌套式菜单”中有嵌套式菜单的详细说明。

菜单命令项目

菜单命令项目可以是文字，也可以是一组由线（分隔线）分隔的一组选择项。分隔线总是用暗淡色彩显示，而且不具有其它特征。

每个菜单命令项目（除分隔线）均有一些明显的特征：

- 菜单命令项目文字左边的图标。如果您要为一个菜单命令项目定义一个图标，可以使用一个图标代表菜单命令项目的文字含意或作用。您可以为一个菜单命令项目指定一个图标、小图标、缩小图标或彩色图标；但是，带有小图标或简化图标的项目不能挂接子菜单，而且不能用当前系统语系以外的其它语系系统进行绘制

- 菜单命令项目文字左边的“✓”号或其它标志符号(项目图标的左边，如果有的话)。如果您需要标志菜单命令项目的状态或它的控制方式，可以使用这种标志。菜单命令项目可以带有标志或子菜单，但不能兼而有之

- 菜单命令项目文字右边用于 Command 键（ ）和其它 1 字符构成的等效命令组合键。使用这种方法，您的应用程序便可让用户从键盘上调用菜单命令，作法是同时按下 Command 键和一个或多个其它的键，就好像用户是从菜单上选择命令一样。带有等效命令组合键的项目不能挂接子菜单、小图标或缩小图标，而且不能用当前系统语系以外的语系进行绘制

- 菜单命令项目文字右边的三角形指示符，表示该项目带有子菜单。带有子菜单的菜单命令项目不能有键盘等效命令、标志符号、小图标或缩小图标，而且不能用当前系统语系之外的语系进行绘制

- 菜单命令项目文字的字形－正常字形或一种其它字形。您可以将菜单命令项目的字形设定为黑体、斜体、下划线、轮廓线、底影或任意一种组合

- 菜单命令项目的文字。选择菜单命令项目的说明文字，用以说明当用户选择命令时将要发生的动作(通常是动词，如打印或存储)。如果该命令可以更改所选对象的属性(例如，黑体或斜体)，您也可以使用形容词。如果没有特别指定，菜单命令项目的文字一般用系统字体和系统字体大小的语系显示（对于罗马语系，系统字体为 Chicago，系统字体大小为 12 点)。如果您不想使用当前系统语系而希望使用另外一种语系来显示菜单命令项目文字的话，您可以指定该文字所用的语系代码。如果安装了指定的语系系统,菜单管理器就可以用语系代码标识的语系来绘制项目文字。用其它语系绘制的菜单命令项目不能挂接子菜单、小图标、或缩小图标

- 在菜单命令项目文字中带有省略号表示在执行命令前，应用程序显示一个对话框来请求用户提供更多的信息。对于显示信息对话框且不需要用户提供附加信息的菜单命令项目，请不要使用省略号。此外，当用户选择了一个菜单命令后，如果应用程序显示一个确认警告栏，您也不应使用省略号。例如，如果用户更改文稿后，选择了关闭命令，您的应用程序可以显示一个确认警告栏，询问用户在关闭前是否要存储该文稿。在此类命令的文字中不应包含省略号

  如果您的应用程序需要显示一个对话框来请求提供更多的信息以响应选择的菜单命令，那么就应该在菜单命令项目文字中包含省略号。例如，在打开命令的文字中应包含省略号，因为用户必须提供附加信息：要打开的文件名。当您在对话框中请求用户提供更多信息时，应在该对话框中提供一个“确定”按钮或等效命令，以便用户选择执行该命令。对话框中还应包含“取消”按钮或等效命令，以便用户取消该命令。请参阅本书“对话管理器”一章中关于创建对话框的说明。

- 暗淡外观显示。当您的应用程序停用一个菜单命令项目时，菜单管理器会淡化该菜单命令项目的色彩，以表示用户不能选择该项目。请注意，菜单管理器淡化显示一切菜单命令项目，包括所有标志或图标、菜单文字、以及任何键盘等效的文字。分隔线始终是暗淡显示的，不论您的应用程序是否启用它们。当您的应用程序停用整个菜单时，菜单管理器将淡化显示该菜单标题和菜单中的所有菜单命令项目

图3－5是两个带有菜单命令项目的菜单，其中有许多您可以用来定义菜单命令项目的特征。

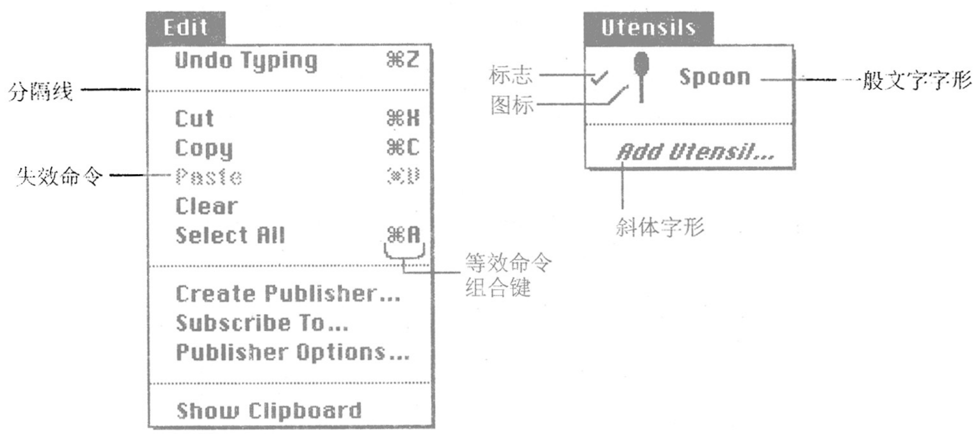


图3-5　两个带有各种特征的菜单

当书写方向是从右到左时（许多非罗马语系系统都属于这种情况，如阿拉伯文等），菜单管理器将反向排列菜单命令项目的各个元素。例如，某些标志符号显示在最右边，某些等效命令组合键显示在菜单命令项目文字的最左边。

在一个被设定成只能显示黑白两种色彩的显示器上，菜单管理器将分隔线显示为点线。对于其它情况，菜单管理器可以根据当前色彩表相应地显示分隔线。例如，在设定为显示4位或更多位彩色的显示器上，菜单管理器一般将分隔线显示成灰色线。

您的菜单中可以包含您希望的任意数目的菜单命令项目。但是，只有前31个菜单命令项目可以被单独停用(第31个菜单命令项目以后的项目总是随该菜单的启用而成为有效项目，随着该菜单的停用而成为无效项目)。如果您的菜单命令项目超过了屏幕的长度，用户必须通过屏幕滚动才能看到其余项目。请记住，菜单中的项目越少，对用户来说菜单也就越简单明了。

菜单命令项目分组

特定菜单中的菜单命令项目在逻辑上应与菜单标题密切相关，而且要进行分组，以便用户更易使用和理解。项目组之间应该用分隔线分开。

一个菜单中可以包含用来执行动作的命令和用来设定属性的命令。您应该使用动词或动词短语来为执行动作的命令命名（例如，剪、复制、贴）。使用形容词为设定所选对象属性的命令命名（例如，黑体、斜体、下划线）。您应该按类别划分菜单命令项目分组：动词（动作）或形容词（属性）。请按照这里所讲述的准则创建各种类型的分组。

分组动作命令在逻辑上是相关的，但各自又是独立的；这样您的菜单便易于读懂。例如，在编辑菜单中，“剪”、“复制”、“贴”、“清除”和“全选”命令为一个分组；创建“发行”、“发行到”和“发行选项”命令为一个分组；“显示剪贴板”命令自己为一个分组。(在3－13页的图3－5上有一个典型的应用程序编辑菜单，其中就有这些命令。)

分组属性命令是相互依赖的。您一般是将一组设定属性的命令分为相互排斥的一组或相互作用的一组。

如果在任意时间，分组中只能有一个属性有效，则该组属性命令便组成一个分组(相互排他组)。应该在当前有效的项目旁边设一个选择标记。如果用户选择分组中的不同属性，可以将选择标记移至新选属性的旁边。例如，图3－6是 SurfWriter 应用程序的彩色菜单。因为不管在什么时候都只能有一种彩色有效，所以该彩色菜单中所列的彩色组成一个互相排斥的分组。在这个实例中，绿色为当前有效色彩。如果用户选择不同的色彩，如蓝色，SurfWriter 应用程序可以使用 SetItemMark 过程将绿色命令旁的检查标记移到蓝色项目旁边。

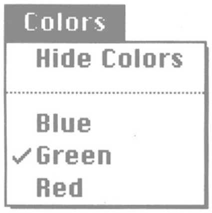


图 3-6　在一个互斥分组中的菜单命令项目

如果一个分组中的多个属性可以同时实现（一个相容分组），那么您也可以将一组属性命令组合起来。在一个相容分组中，可以用选择标记来表示多重属性有效。在这类分组中，您还需要提供一个取消其它所有属性的命令。例如，允许用户任意选择字形组合的字形菜单，应该也包括一个“正常”字形，它可以取消其它所有属性。图 3-7 是一个字形菜单；在这个例子中，“黑体”和“空心”属性有效。

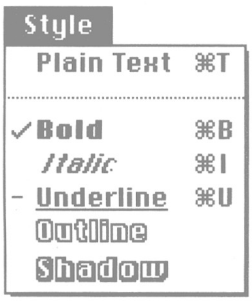


图 3-7　在一个相容分组中的菜单命令项目

您也可以使用选择标记和破折号的组合来帮助指示用户内容的状态。例如，在一个反映选定项状态的菜单中，若属性适用于全选项，可以将一个选择标记画在项目的旁边，若属性仅适用于一部分选定项，则可以在项目旁边画一个破折号。图 3-8 显示了一个菜单，它表示选定项有一种以上的字形。在该图中，黑体属性适用于全部选定项，而下划线选项只适用于部分选定项。



图 3-8　在一个相容分组中使用选择标记和破折号

在显示应用程序的菜单前，您的应用程序应适当调整其菜单。例如，您应该根据用户文稿的状态和当前窗口的类型，为必要属性的项目添加选择标记或破折号。详细信息请参阅 3-73 页的“调整应用程序的菜单”。

使用切换命令是另一种显示属性存在或不存在的方法。如果属性有两种状态，而您希望用户使用一个菜单命令在两种状态间切换，那么就可以使用切换命令。例如，在文稿中，如果没有显示环绕版本和引用程序的边框，您的应用程序就会提供一个“显示边框”的命令。当用户选择了“显示边框”命令后，您的应用程序应该显示边框并将菜单命令项目改成“隐藏边框”。当用户选择了“隐藏边框”命令后，您的应用程序应该隐藏所有版本或引用程序的边框，同时将菜单命令项目改变成“显示边框”。只有当命令的两种用词不会使用户感到迷惑时，才能使用切换命令。您应该选择动词短语作为切换命令的文字；而且文字应该清楚地表明当用户选择该项目时，您的应用程序将执行的动作。有关切换命令的进一步信息，请参阅 3-59 页的“更改命令项目文字”。

菜单命令的等效命令组合键

每个菜单命令可以有一个等效命令组合键。等效命令组合键这个术语是指一个键盘组合，例如 Command — C（— C）或者 Command 键、其它键和一个或多个修正键的任何组合。当用户按下该组合键时，请求执行相应的菜单命令。例如，若您的应用程序支持文件菜单中的“新文稿”命令，您的应用程序应该在用户按 “Command” ＋ “N” 组合键时，执行用户在文件菜单中选择“新文稿”时同样的动作。等效命令组合键这个术语特指用户按住 Command 键，同时按下另一个键（修正键除外）来请求执行相应的等效命令组合键。这样就生成了一个键盘事件，它指定了一个 1 字节的字符，您的应用程序应该将它作为参数传送给 MenuKey 函数。MenuKey 函数将给定的 1 字节字符用该等效命令组合键映射到菜单命令项目（如果有的话）。

菜单管理器支持等效命令组合键。若您为一个菜单命令项目定义了等效命令组合键，标准菜单定义过程将绘制 Command 字符，并在菜单命令项目文字的右边指定一个的字符（如果书写方向是从右向左，就在项目文字的左边指定）。

您可以通过考察键盘事件的事件记录中的 modifiers 域来检测一个命令的等效命令组合键。这将使您能够确定在发生键盘事件的同时，Command 键是否按下。若 Command 键被按下，您的应用程序通常就会调用 MenuKey 函数，将代表用户所按键的字符代码作为一个参数来传送。MenuKey 函数能够确定该字符是否符合菜单命令项目定义的某个等效命令组合键；如果符合，MenuKey 便将该信息返回您的应用程序。然后，您的应用程序可以执行有关的菜单命令（如果有的话）。有关事件记录 modifiers 域的附加信息，请参阅本书的“事件管理器”一章。

当 Command 键按下时，一些键盘的键盘布局（'KCHR'）资源会屏蔽或取消 Shift 键的作用。例如，对一个美国键盘，当用户按下“Command” ＋ “S” 键组合时，在事件记录的消息域中的字符代码是 $73（“s” 的字符代码）；当用户按 Command — Shift

—S 键组合时，事件记录的消息域的字符代码仍是 $73。当然，并非所有的 'KCHR' 资源都以这种方式屏蔽 Shift 键。

而且，当您的应用程序使用 MenuKey 函数来处理等效命令组合键时，MenuKey 并不区分大小写字母。MenuKey 函数接收传送给它的那一个字符，并调用 UpperText 过程（它对字符进行本地化大写转换）。这样，MenuKey 在将键盘事件和等效组合键比较时，会将任何小写字符转换成大写。如果您的应用程序必须区分等效命令组合键的大小写字符，那么您就需要自己提供处理这类等效命令组合键的方法。

您为等效命令组合键指定的键必须是一个 1 字节的字符，通常是一个字母（虽然您也可以指定字母以外的 1 字节字符）。为了保持应用程序的一致性以及为应用程序的本地化提供较大的支持，在定义应用程序菜单命令时，应该总是将大写字母指定为等效命令组合键。

如果您希望提供等效命令组合键之外的其它类型的等效命令组合键，那么您的应用程序必须采取附加步骤来支持它们。如果您的应用程序允许用户按下一个以上的修正键来请求执行一个等效命令组合键，那么您的应用程序必须在菜单命令项目中提供一个代表该键盘组合的可视的指示符。在大多数情况下，您的应用程序必须使用它自己的方法（不同于 MenuKey）将等效命令组合键映射到相应的菜单命令项目中。

如果您为等效命令组合键指定一个非字母键，或使用一个以上的修正键作为等效命令组合键，您应该选择容易为其它地区本地化的键或键盘组合。

如果您的应用程序使用其它的等效命令组合键，您可以考察修正键的状态，并在必要时使用KeyTranslate 函数来帮助将等效命令组合键映射到特定的菜单命令项目上。有关 KeyTranslate 函数请参阅本书的“事件管理器”一章，有关各种键盘组合如何映射到指定的字符代码，请参阅《Inside Macintosh 文本》（*Inside Macintosh: Text*）一书中关于 'KCHR' 资源的讨论。

一个未列在菜单中但可以从键盘请求的命令是“Command” + “·”或“取消”命令。您可以使用类似于检测其它等效命令组合键的方法检测“Command” + “·”命令，即您通过检验一个键盘事件的 modifiers 域来确定 Command 键是否按下。但是，在这种情况下，若用户同时按下了“·”键，您的应用程序就应该取消当前的操作，而不是请求执行一个菜单命令。

通常您是在一个 'MENU' 资源中定义菜单命令时，定义应用程序菜单命令的 Command 等效组合键的。菜单管理器在菜单命令项目文字右边（对于书写为从右到左的语系系统在项目文字的左边）显示菜单命令的等效命令组合键（如果有的话）。

苹果公司为通用的命令保留了几个等效命令组合键。您应该在应用程序的文件和编辑菜单中使用这些等效命令组合键。

表 3-1 显示了标准命令的等效命令组合键。

表 3-1　为所有系统保留的等效命令组合键

键	命令	菜单
⌘-A	“全选”	编辑
⌘-C	“复制”	编辑
⌘-N	“新文稿”	文件
⌘-O	“打开”	文件
⌘-P	“打印”	文件
⌘-Q	“退出”	文件
⌘-S	“存储”	文件
⌘-V	“贴”	编辑
⌘-W	“关闭”	文件
⌘-X	“剪”	编辑
⌘-Z	取消”	编辑

附注：

为了支持桌面附件和对话框中的编辑，您应该对编辑命令“取消”、“剪”、“复制”和“贴”分别使用等效命令组合键 Z，X，C 及 V。

苹果公司还为世界范围的系统软件版本、本地化键盘和键盘布局保留了几个等效命令组合键。表 3-2 显示了这些等效命令组合键。您的应用程序不应该将表 3-2 中列出的等效命令组合键用作自己的菜单命令。

有关在其它语系系统中处理等效命令组合键的详细讨论，请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书。

表 3-1 和 3-2 中列出的键组合对所有的应用程序都是保留的。即使您的应用程序不支持其中一个菜单命令，您也不应该将它用作其它命令的等效命令组合键。这个准则是为了用户的利益。保留这些键组合能够对所有的应用程序提供有保证的、可预期的行为。

表 3-2 为世界范围的系统保留的等效命令组合键

键	动作
⌘- 空格键	在启用的语系系统中轮换
⌘-Option 一空格键	在激活的语系系统中轮换键盘布局或输入方法
⌘- 修正键一空格键	保留
⌘- 右箭头键	将键盘布局改变成 罗马 字体的当前键盘布局
⌘- 左箭头键	将键盘布局改变成系统字体的当前键盘布局

表3-3显示了其它常用的等效命令组合键。这些等效命令组合键没有表3-1和表 3-2 中列出的那些标准等效命令组合键重要。如果您的应用程序不支持表 3-3 中的某个功能，您可以根据自己的需要使用该等效命令组合键。

表 3-3 其它常用的等效命令组合键

键	命令	菜单
⌘-B	黑体	字形
⌘-F	查找	文件
⌘-G	再次查找	文件
⌘-I	斜体	字形
⌘-T	简明字体	字形
⌘-U	下划线	字形

您不应该为不常用的菜单命令指派等效命令组合键。只应为用户使用最频繁的那些命令添加等效命令组合键。

菜单管理器自动添加菜单

在 System 7 中，菜单管理器最多可以为您的应用程序添加 三 个辅助菜单：帮助菜单、键盘菜单和应用程序菜单。这些菜单允许用户存取系统功能，例如旁述、键盘布局和应用程序切换。所有这 三个菜单均采用图标作为标题，菜单位置在菜单栏的右边。(有时，这些菜单是指系统处理菜单。)

当您的应用程序将一个苹果菜单插入自己的菜单栏时，菜单管理器会自动将这些辅助菜单插入应用程序的当前菜单列表中。这时，菜单管理器总是显示应用程序菜单。如果空间许可，也显示帮助菜单；如果安装了多种语系系统并且空间许可，也显示键盘菜单；如果设定了 ‘itlc’ 资源的标志字节中的 smfShowIcon 位，菜单管理器也显示键盘菜单。

如果您的应用程序插入了一个菜单，而且标题延伸到了帮助菜单的图标空间或帮助菜单和键盘菜单两者的图标所占据的空间，那么帮助菜单的图标或帮助菜单和键盘菜单两者的图标都会从菜单栏中消失。这使得您的应用程序能够获得原来被这两个或其中之一的菜单图标占据的菜单栏空间。但是，如果您的应用程序插入了一个菜单，而其标题较长，以致于和应用程序菜单图标占据的空间重叠，那么该标题的重叠部分将消失在应用程序菜单图标的后面。应用程序菜单图标总是显示在菜单栏中。

因为菜单管理器将帮助、键盘和应用程序菜单插入了应用程序的当前菜单栏，所以您不应该对菜单栏的上一个或多个菜单作任何假定。苹果公司保留为您的应用程序的菜单栏添加其它系统处理菜单的权利；出于兼容性的考虑，您的应用程序菜单栏的定义应该为帮助、键盘、应用程序以及至少一个辅助系统处理菜单留出空间。

如果用户从键盘或应用程序菜单选择了一个项目，那么您的应用程序就不需要采取任何动作；菜单管理器会对这两个菜单执行任何适当的动作。若用户选择了一个您的应用程序添加到帮助菜单中的项目，您的应用程序就应该采取相应的动作。

以下各节详细介绍了帮助、键盘和应用程序菜单，而且还介绍了一个典型应用程序中的其它菜单，包括苹果、文件和编辑三个标准菜单。

苹果菜单

您应该在应用程序中将苹果菜单定义为第一个菜单。苹果菜单的标题是苹果图标。一个应用程序典型的苹果菜单的第一个菜单命令项目是“关于…”命令，下面是一条分组线，再下面是苹果菜单命令项目文件夹中包含的桌面对象的列表。(桌面对象是指应用程序、桌面附件、文稿、文件夹和任何其它能够放在苹果菜单项目文件夹中的项目。)苹果菜单中分组线以下的项目按字母顺序列出。分组线下面的每一个项均列出了一个桌面对象和该对象的小图标。

图3-9是SurfWriter应用程序的苹果菜单在一个特定的用户系统中可能出现的形式。

为了在您的应用程序的苹果菜单中创建项目，就需要在一个 'MENU' 资源中定义苹果菜单标题、您的应用程序的“关于…”命令的特征以及它下面的分组线。

为了将包含在苹果菜单命令项目文件夹中的项目插入应用程序的苹果菜单，需要使用 AppendResMenu 或 InsertResMenu 过程，并指定 'DRVR' 作为资源类型添加在参数 theType 中。这样做后，这些过程会自动地按照字母顺序将所有在苹果菜单项目文件夹中的项目添加到指定菜单中。



图 3-9　SurfWriter 应用程序的苹果菜单

附注:

在 System 7 及其更高版本中都具有苹果菜单命令项目文件夹。在 System 6 中，当您指定 'DRVR' 作为资源类型添加到参数 theType 中时，AppendResMenu 和 InsertResMenu 过程只将系统文件中的桌面附件添加到指定的菜单中。

用户可以将任何桌面对象放入苹果菜单命令项目文件夹。当用户在该文件夹中放入一个项目后，系统软件便会自动将它添加到所有打开的应用程序的苹果菜单命令项目列表中。

当用户从苹果菜单中选择一个项目（不是您的应用程序的“关于...”命令）时，您的应用程序应该调用 OpenDeskAcc 函数。OpenDeskAcc 函数时刻准备打开用户选择的桌面对象；例如，用户选择闹钟桌面附件，OpenDeskAcc 就准备打开闹钟。OpenDeskAcc 函数负责为闹钟桌面附件安排执行时间，然后返回您的应用程序。在您的应用程序下一次调用 WaitNextEvent 时，它会接收到一个挂起事件，然后闹钟桌面附件就会变成前台进程。

如果用户选择了一个桌面对象（不是一个桌面附件）或者一个应用程序，OpenDesk-Acc 函数也采取适当的动作。例如，如图 3-9 所示，若用户获取了一个由 SurfWriter 应用程序创建的名为 MyTideReport 的文稿，OpenDeskAcc 函数则准备打开 SurfWriter 应用程序（如果它尚未打开），并准备执行 SurfWriter 应用程序。当 SurfWriter 应用程序成为前台进程后，即指示它打开 MyTideReport 文稿。

当用户选择您的应用程序的“关于...”命令时，您的应用程序可以显示一个对话框或一个警告栏，它包含您的应用程序的名称、版本号、版权信息或其它必要的信息。您的应用程序应该在对话框中提供一个“确定”按钮；用户单击该“确定”按钮可以关闭该对话框。

图 3-10 显示了当用户从应用程序的苹果菜单中选择了“关于 ...”命令后，SurfWriter 应用程序显示的警告栏。

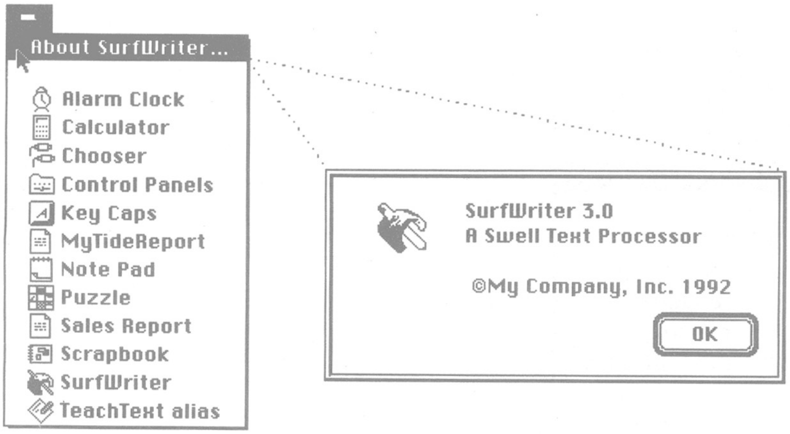


图 3-10　选择 SurfWriter 应用程序的“关于 ...”命令

如果您的应用程序提供了应用程序专有的“帮助”命令，那么就应该将它们放在帮助菜单，而不是苹果菜单中。

文件菜单

标准的文件菜单包含与文件管理有关的命令。例如，用户能够从这个菜单打开、关闭、存储或打印文稿。用户还应该能够从文件菜单中选择“退出”命令，以便从您的应用程序中退出。

您的应用程序应该支持标准文件菜单的菜单命令。如果您为应用程序的文件菜单添加其它的命令，那么所添加的命令应该是管理文稿的命令。

图 3-11 是一个应用程序的标准文件菜单。

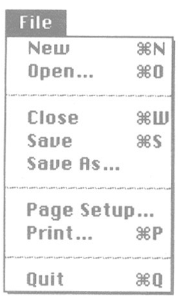


图 3-11　一个应用程序的标准文件菜单

表 3-4 列出了文件菜单中的标准命令，以及当用户获取这些命令时，您的应用程序应该采取的动作。

表 3-4 响应标准文件菜单命令的动作

命令	动作
"新文稿"	打开一个新的无标题文稿
"打开"	使用标准文件包显示打开对话框
"关闭"	关闭当前活动窗口(它可能是一个文稿窗口，不定模式对话框或其它类型的口)。若当前活动窗口是一个文稿窗口，而且文稿从上次存储以来已经更改了，那么将会显示一个对话框，询问用户在窗口关闭之前，是否要存储文稿。
"存储"	将当前文稿（包括上次存储以来对文稿所作的任何改变），存入一个磁盘。如果用户选择"存储"命令，用于一个新的未标题文稿（即用户尚未命名的文稿），那么就会使用标准文件包显示一个存储对话框。
"存储为"	将当前文稿按照用户提供的新名称存储一个拷贝。使用标准文件包显示存储对话框。在您的应用程序存储文稿后，文稿应该仍然是打开并激活的。
"页面设置"	显示"页面设置"对话框，让用户指定打印参数，例如页面大小和打印方向。您的应用程序可以提供其它适当的打印选项。当用户存储文稿时，您的应用程序应该存储用户对文稿的"页面设置"打印参数选择。
"打印"	显示"打印作业"对话框，让用户指定各种参数，例如打印质量和份数。用户点按"打印"按钮即可打印文稿。打印对话框中指定的选项只适用于当前的打印操作，您的应用程序不应该将它们和文稿一起存储或在用户再次选择"打印"时还原设定状态。
"退出"	在执行任何必要的清除过程后，退出您的应用程序。若某个打开的文稿自用户上次存储它们以来被更改过，则应对需要存储的打开文稿显示一次存储对话框。若有任何后台或冗长的操作仍在进行，将给用户一个继续执行的选择。

有关您可以在文件菜单中提供的附加命令，请参阅《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书。关于如何执行和文件菜单中命令有关的动作的信息，请参阅《Inside Macintosh 文件》(*Inside Macintosh: File*) 一书的"文件管理介绍"一章。关于标准文件对话框的信息，请参阅《Inside Macintosh 文件》(*Inside Macintosh: File*) 一书中"标准文件包"一章。关于显示页面设置和打印作业对话框的信息，请参阅《Inside Macintosh 绘图》(*Inside Macintosh: Imaging*) 一书中"打印管理器"一章。

"新文稿"、"打开"、"关闭"、"存储"、"打印"和"退出"命令的等效命令组合键显示在 3-22 页中的表 3-11 中。那些等效命令组合键都是为这些菜单命令保留的;不要将那些等效命令组合键指派给图 3-11 中没有显示的菜单命令。

编辑菜单

标准编辑菜单为用户提供了更改或编辑文稿内容的命令。另外还提供了使用户能够使用编辑信息或剪贴板在文稿内或不同应用程序创建的文稿间共享数据的命令。所有 Macintosh 应用程序都应支持"取消"、"剪"、"复制"、"贴"和"清除"命令。您可以使用这些命令在您的应用程序中提供标准的文字编辑能力。

图 3-12 显示了 Macintosh 应用程序支持的标准编辑菜单。

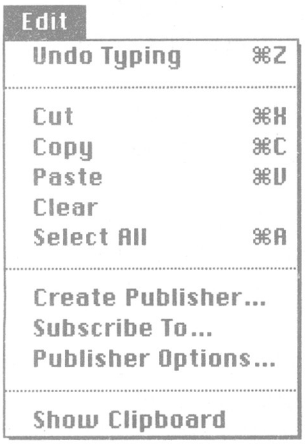


图 3-12 一个应用程序的标准编辑菜单

您的应用程序编辑菜单中的标准编辑命令("取消"、"剪"、"复制"、"贴"和"清除")应该以图 3-12 显示的次序出现。只要可能,您应该添加一个附加的词或短语来说明:当用户选择"取消"命令时,您的应用程序的什么动作将被取消。例如,图 3-12 显示了一个应用程序的编辑菜单,当键入用户执行的最后一个动作时,它使用短语"取消键入"。若您的应用程序不能取消最后一次操作,您应该将"取消"命令的文字改变为"不能取消"并停用该菜单命令项目。有关如何更改菜单命令项目文字的实例,请参阅 3-59 页中"更改项目文字"。

您的应用程序的编辑菜单中可以包括其它命令,只要它们和编辑或更改您的应用程序的文稿内容有关就行。如果您需要给编辑菜单添加命令,应添加在标准菜单命令之后。例如,如果适用,您的应用程序应该支持"全选"命令。若您的应用程序既支持"清除"也支持"全选",则应按照图 3-12 中的次序出现。

表 3-5 列出了编辑菜单的标准命令以及当用户选择后您的应用程序应该采取的动作。

表 3-5　针对标准编辑菜单命令的动作

命令	动作
"取消"	取消上次操作的影响。您应该为"取消"命令添加上次操作的名称。例如，如果用户刚刚结束了在一个文稿中文字输入工作，那么就应该将项目改变为"取消键入"。如果您的应用程序不能取消上次操作，就应该停用该项目，并将项目短语改为"不能取消"。
"剪"	若有的话，可以在当前选定项中删除数据。将剪下的选定项数据储存在剪贴数据存储区(剪贴板)。这些数据将会代替剪贴数据存储区中原来容。
"复制"	若有的话，可以在当前选定项中拷贝数据。将复制的选定项数据储存在剪贴数据存储区（剪贴板）。它代替剪贴数据存储区中原来的内容。
"贴"	将剪贴数据存储区的数据贴在插入点；代替当前的选定项。
"清除"	删除醒目显示选定项的数据。不要将数据拷贝到剪贴数据存储区（剪贴板）。
"全选"	醒目显示文稿中的全部数据。
"创建发行"	显示"创建版本"对话框(使用版本管理器)。若用户单击"程序"、"出版"按钮，根据选择的数据创建一个编辑信息。
"订阅"	显示"订阅"对话框（使用版本管理器)。若用户单击"订阅"按钮，即可从一个版本插入数据。
"版本显示"	"版本选项"对话框（使用版本管理器)，并允许用户选项、设定或更改和版本有关的选项。如果当前选定项包括一个引用程序，则可以将该菜单命令项目改变为"引用程序选项"。当用户选择"引用程序选项"　命令时，将显示"引用程序选项"对话框。
"显示剪贴板"	在一个窗口中显示剪贴板的内容。当剪贴板窗口显示时，将该项目改变为"隐藏剪贴板"。当用户选择"隐藏剪贴板"时，隐藏显示剪贴板内容的窗口并将菜单命令项目改变为显示剪贴板。

"取消"、"剪"、"复制"、"贴"和"全选"命令的等效命令组合键显示在 3-24 页中的 图 3-12。那些等效命令组合键是为这些菜单命令保留的；不要将那些等效命令组合键指派给图 3-12 没有显示的任何菜单命令。有关将数据拷贝到剪贴数据存储区及从它拷贝数据的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中"剪贴数据存储区管理器"一章。

有关在您的应用程序中支持“创建发行物”、“订阅”和“发行选项”命令的信息，请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书中“发行管理器”一章。

字体菜单

您可以为用户提供一个字体菜单，以使用户选择文形。一种字体是以字体和字形创建的一组完全的字符。一种字体的字符可以有不同大小，但是，它们有相同的设计元素。

您应该在应用程序的字体菜单中列出所有当前可用字体的名称。当前可用字体是在用户系统文件夹（在较早版本的系统软件中，它在用户的系统文件中）的字体文件夹中的那些字体。

您可以使用 AppendResMenu 或 InsertResMenu 过程为字体菜单添加字体。而且，这两个过程可以按字母顺序将项目添加到指定的菜单中。

用户可以安装多种字体，创建一个很大的字体菜单。因此，您不应该在字体菜单中包括其它项目。应使用另外的菜单来包含属性列表，例如“字形”和“大小”的选定项。您也可以提供一个大小菜单，使用户能够选择指定大小的字体；下一节介绍大小菜单。

图 3-13 显示了一个典型的字体菜单。您的应用程序应该通过在当前字体名称的左边添加选择标记表明那一种字体正在被使用。在图 3-13 中，应用程序在Palatino的旁边画了一个选择标记，表明 Palatino 是当前字体。当用户开始在插入点输入文字时，您的应用程序应该以当前字体显示文字。



图 3-13 一个典型的字体菜单

在字体菜单中，您可以使用破折号来表明选定项包含了一种以上的字体。（若整个选定项只包括一种字体，在项目的旁边画一个检查标记。）若当前选定项包含了一种以上的字体，在选定项所包含的每一种字体旁边画一个破折号。有关为菜单命令项目添加破折号和选择标记的信息，请参阅 3-61 页的“更改菜单命令项目的标记”。

图 3-14 显示了使用破折号来表明在一个选定项中包含了一种以上的字体。在该图中，一部分选定项包含 Helvetica 字体，一部分选定项包含 Palatino 字体。



图 3-14 一个显示了选定项包含一种以上字体的字体菜单

AppendResMenu 和 InsertResMenu 过程可以识别出添加的字体资源是否属于当前系统的语系（非罗马字体的字体号码大于 $4000)。若安装了某种字体的语系系统，菜单管理器将在其相应的语系中显示该字体的名称。

除了字体菜单外，您还可以选择提供大小菜单和字形菜单。若这样做，这三种菜单在大多数应用程序中一般以字体、大小和字形的次序出现。

大小菜单

您的应用程序可以提供一个大小菜单，使用户能够选择字体的大小。字体大小用点数测量。点是测量的单位，在 Macintosh 电脑上等于 1/72 英寸。

您的应用程序应该通过在菜单命令项目上的当前大小旁边添加选择标记，来表明字体的当前的点数。若选定项包含了一种以上的大小，您可以使用破折号。

System 7 既支持点阵字体也支持 TrueType 字体。TrueType 字体可以显示较大范围的点数，例如，12 点、51 点、156 点、578 点或者更大。您的应用程序不应该对字体大小给出上限。

在大小菜单中，您的应用程序应该为字体大小加轮廓线，以表明当前字体直接提供的大小种类。若用户选择 TrueType 字体，应该在大小菜单中为所有的字体大小加轮廓线。若用户选择一个点阵字体，只需为那些出现在字体文件夹中的字体大小加轮廓线。对于所有其它字体大小，均使用正常字形。有关您的应用程序支持字体的附加信息，请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书中“字体管理器”一章。

图 3-15 是一个应用程序典型的大小菜单。

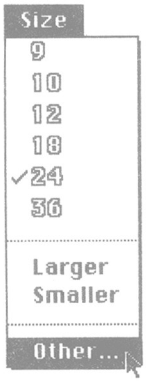


图 3-15 一个典型的大小菜单

您的应用程序也应该提供一种方法，使用户能够选择任意大小的点数。为此，您可以在大小菜单的最后添加一个“其它”命令项目。当用户选择该命令时，将显示一个对话框，使用户可以选择任何可用的字体大小。您可以为该对话框设一个可编辑的文字项目，以便用户在其中键入需要的字体大小。图 3-16 是一个对话框，当用户在大小菜单中选择“其它”命令时，应用程序可以显示这个对话框。

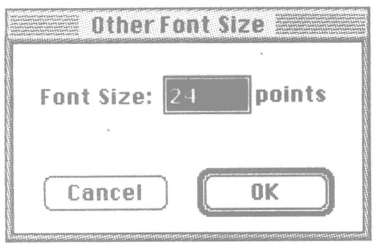


图 3-16 一个选择新点数字体的对话框

图 3-17 显示了一个其它对话框，用户输入一个点数为 31 的新字体大小。

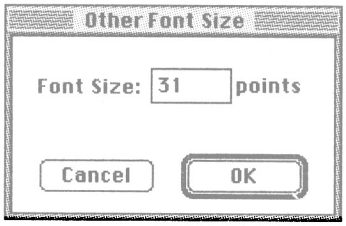


图 3-17　输入一个新的字体大小

若用户输入了一个不在当前菜单中的字体大小，您的应用程序应该为其它“命令”添加一个选择标记，并将该字体大小作为“其它”命令文字的一部分。您应该如图 3-18 所示，在文字“其它”后面的括号中显示该字体大小。

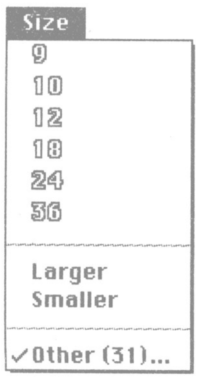


图 3-18　带有一种添加字体大小的“其它”命令

若选定项包含了一种以上的非标准字体，您应该在“其它”后面的括号内包括文字“混合”。这时，当用户选择“其它”（混合）命令时，令“其它”对话框的可编辑文字域为空白。

有关如何响应用户从大小菜单中选择命令的更详细的信息，请参阅 3--82 页的“处理大小菜单”。有关创建对话框的信息，请参阅“对话管理器”一章。

帮助菜单

帮助菜单就像苹果、文件和编辑菜单一样，是每个应用程序专有的。帮助管理器定义的帮助菜单命令项目对所有的应用程序是通用的，并使用户能够存取旁述。

您可以为应用程序的帮助菜单添加菜单命令项目，使您的用户能够除了存取旁述外，还可以存取您的应用程序提供的任何联机帮助。若您当前为用户提供帮助信息的方法是用户选择苹果菜单中的“关于…”命令，那么您应该为帮助菜单附加一个您自己的应用程序的帮助命令。这样，就为用户提供一个获得帮助信息的理想途径。

在为帮助菜单添加您自己的项目时，应该在命令中包括您的应用程序的名称，以便使用户能够容易地确定“帮助”和哪一个应用程序有关。

图 3-19 显示了 SurfWriter 应用程序的帮助菜单。这个应用程序在标准帮助菜单最下面附加了一个项目： SurfWriter 帮助。当用户选择该项目时，应用程序将允许用户存取该应用程序的专门帮助信息。



图 3-19 SurfWriter 应用程序的帮助菜单

您可以使用 HMGetHelpMenuHandle 函数和 AppendMenu 过程来为帮助菜单添加项目。苹果公司保留在帮助菜单中更改标准项目数的权利。您应该总是将任何附加的项目添加在最下面。具体实例，请参阅 3-67 页的“为帮助菜单添加命令项目”。

用户通过选择帮助菜单中的显示旁述和隐藏旁述来开关旁述。当用户在帮助菜单中选择“显示旁述”或“隐藏旁述”时，帮助管理器可以自动启用或停用旁述。帮助的设定状态是全局的，将影响所有的应用程序。

在用户打开旁述后，帮助管理器会随着光标在窗口或对话框的滚动杆、按钮、菜单中的矩形区域（如果有帮助信息）内移动时，显示小型旁述。旁述是圆角矩形窗口，它可以为用户提供说明信息。

帮助管理器对苹果、帮助、应用程序和键盘菜单的菜单标题提供了旁述。帮助管理器还对应用程序和键盘菜单中的菜单命令项目、苹果菜单的苹果菜单命令项目文件夹中的所有项目、以及帮助菜单中的标准项目提供了旁述。只有当您的应用程序使用标准的菜单定义过程时，帮助管理器才提供这些旁述。

您应该为您的应用程序中所有其它的菜单命令项目和菜单提供旁述的内容。

图 3-20 显示了苹果菜单标题和应用程序菜单标题的系统设定旁述。



图 3-20　苹果菜单和应用程序菜单的系统设定旁述

图 3-21 是应用程序的“剪”命令启用和停用时的旁述。

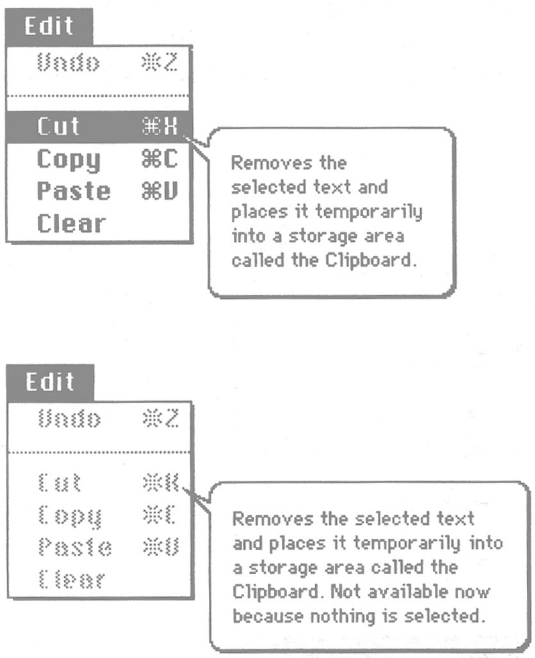


图 3-21　不同状态时“剪”命令的旁述

您的应用程序能够为您的菜单和菜单命令项目提供旁述的内容。您可以使用‘hmmu’资源为您的应用程序定义旁述。

有关如何在‘hmmu’资源中定义应用程序菜单的旁述的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Essentials*) 一书的“帮助管理器”一章。

键盘菜单

键盘菜单显示了一个键盘布局和输入方法的列表。每个启用的语系系统都可以使用这个列表。每个语系系统至少有一个自己的键盘布局或输入方法。如果只能使用罗马语系系统和美国键盘布局，那么菜单管理器就不会添加键盘菜单（除非 ‘itlc’ 资源的标志字节中的smfShowIcon 位被设定）。若用户的系统包括一个辅助语系系统或包括罗马语系系统附加的键盘布局，并且 ‘itlc’ 资源中的 smfShowIcon 位被设定，您应用程序的菜单栏又包括一个苹果菜单，菜单管理器才会将键盘菜单添加到您的应用程序的菜单栏中。菜单管理器将把键盘菜单添加到帮助菜单的右边，苹果菜单的左边。

图 3-22 显示了一个可能在特定的用户系统中出现的键盘菜单。系统软件按照键盘菜单命令项目所属的语系系统对它们进行分组。例如，图 3-22 中有 7 种语系系统：阿拉伯、罗马、西里尔、希伯莱、泰文、日文、韩文。用户系统中的阿拉伯语系系统可以使用两种键盘布局，罗马语系系统可以使用两种键盘布局，西里尔语系系统可以使用一种键盘布局，希伯莱语系系统可用两种键盘布局，泰文语系系统可以使用三种键盘布局，日文语系系统有两种输入方法，韩文语系系统有一种输入方法。

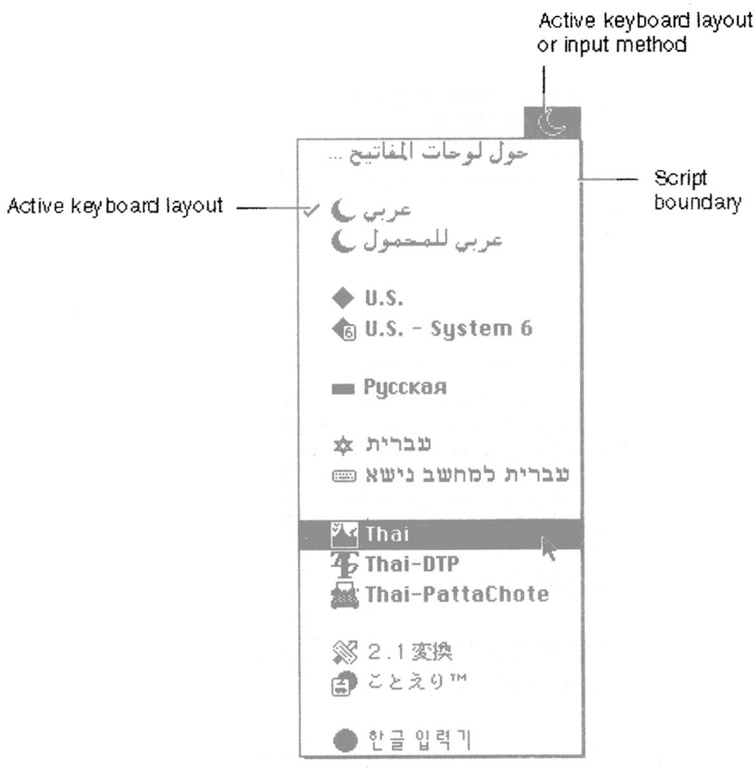


图 3-22　在一个应用程序中存取键盘菜单

当用户从键盘菜单中选择一个项目时，菜单管理器会适当地处理它。例如，若用户选择了一个不同语系系统中的键盘布局，菜单管理器会将当前键盘布局和语系系统改变为用户选择的键盘布局和语系系统。有关在应用程序的多种语系系统中支持文字和处理文字的更详细的信息，请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书。

应用程序菜单

应用程序菜单是菜单栏最右边的菜单，应用程序菜单包含当前应用程序或桌面附件的图标作为它的菜单标题。

如果您的菜单栏包含一个苹果菜单，菜单管理器就会自动地将应用程序菜单附加到您应用程序的菜单栏。

当用户从应用程序菜单选择一个项目时，菜单管理器也会适当地处理这个事件。例如，若用户选择"隐藏其它"命令，菜单管理器隐藏所有其它打开的应用程序的窗口。若用户从应用程序菜单选择另一个应用程序，菜单管理器便向您的应用程序发送一个挂起事件。您的应用程序在下一次调用 WaitNextEvent 时，将接收该挂起事件，您的应用程序在处理了这个挂起事件后即切换出去。（有关响应挂起事件和恢复执行事件的信息，请参阅本书的"事件管理器"一章。）

图 3-23 显示了SurfWriter应用程序的应用程序菜单，这时SurfWriter 和TeachText都打开了，用户当前正和 SurfWriter 对话。在显示 SurfWriter 图标的菜单命令项目旁边的选择标记表明 SurfWriter 是激活的应用程序。



图 3-23 SurfWriter 的应用程序菜单

弹出式菜单

您可以使用弹出式菜单在一个对话框或窗口中给用户一个选择列表。弹出式菜单在要求用户从许多选择中选定一种或设定一个特殊值的对话框中特别有用。

在 System 7 中，标准的弹出式菜单是用一个控制部件定义函数执行的。本节阐述了标准弹出式控制部件定义函数是如何为弹出式菜单提供支持的。本书的"控制管理器"一章对控制部件作了详细的解释。

弹出式菜单的外观是一个矩形，它的边框宽度是一个像素，阴影宽度也是一个像素。弹出式菜单用一个弹出方框中向下的三角形来标识。弹出式菜单的标题出现在弹出方框的左边。图 3-24 显示了一个弹出式菜单。

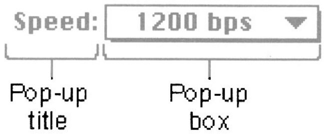


图 3-24　一个弹出式菜单

当光标在弹出标题或弹出方框上时，用户按鼠标按钮即可显示弹出式菜单。若弹出式菜单在一个对话框中，而且您的应用程序使用对话管理器，对话管理器将使用弹出控制部件定义函数来显示弹出式菜单，并处理用户所有在弹出式菜单中的交互作用。若弹出式菜单在您的应用程序的一个窗口中，您的应用程序需要确定当用户按下鼠标按钮时，光标在那个控制部件中。然后，您的应用程序可以使用控制管理器例程来显示弹出式菜单并在该控制部件中处理与用户的交互作用。

和 MenuSelect 一样，弹出式控制部件定义函数醒目显示弹出式菜单标题，并在用户拖拽光标通过菜单命令项目时，适当地醒目显示其菜单命令项目。当弹出式菜单首次显示时，弹出式控制部件定义函数还醒目显示系统设定（当前）的菜单命令项目，并为Macintosh 用户界面工具箱添加选择标记。用户一释放鼠标按钮，弹出控制部件定义函数就会闪烁所选的项目（如果有的话）、取消菜单标题的醒目显示、更改弹出方框中的文字并将选择的项目数作为控制值储存。您的应用程序可以使用控制管理器函数 GetControlValue 来得到用户选择的菜单命令项目。

图 3-25 显示了关闭状态的一个弹出式菜单（它向用户展示的初始外观）及它打开的状态（当光标在弹出式菜单中，用户按下鼠标按钮后展示的外观）。



图 3-25　一个弹出式菜单的关闭和打开状态

如果您不为一个弹出式菜单提供标题，那么当前菜单命令项目将作为该菜单的标题。在大多数情况下，您应该建立一个有标题的弹出式菜单。应选择一个能够反映菜单内容或指示菜单用途的标题。

图 3-26 显示了用户在弹出式菜单中进行选择的过程。

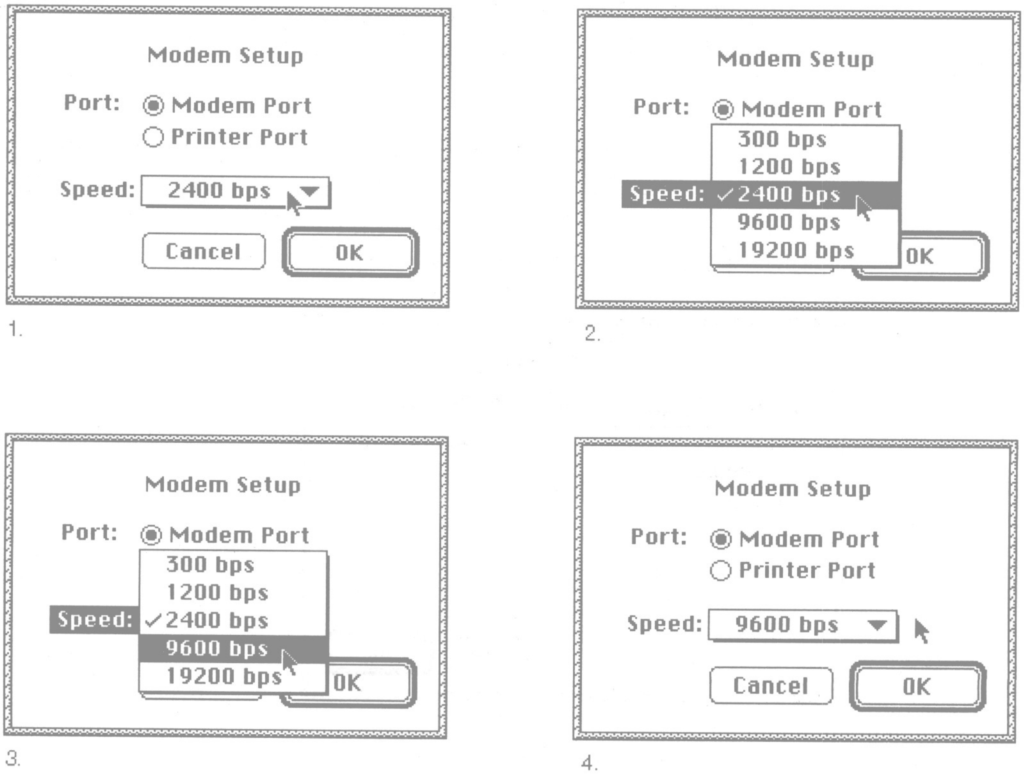


图 3-26　在弹出式菜单中进行选择

在图 3-26 的第一步中，当光标在弹出方框上时，用户按下鼠标按钮。这时，您的应用程序可以使用对话管理器或控制管理器来调用弹出控制部件定义函数。在第二步中，弹出控制部件定义函数醒目显示弹出式菜单的标题，从弹出方框中取消向下的三角形，为当前项目添加一个选择标记，醒目显示当前项目，并显示弹出式菜单的内容。在第三步中，推出控制部件定义函数处理与用户的所有交互作用，醒目显示菜单命令项目或取消它们的醒目显示，直至用户释放了鼠标按钮。当用户释放了鼠标按钮后，推出控制部件定义函数关闭弹出式菜单，取消弹出式菜单标题的醒目显示，将弹出方框的文字设定到用户选择的项目，并将选择的项目数作为控制值储存。第四步显示了弹出控制部件定义函数执行了这些动作后，关闭时弹出式菜单的观。

若您的应用程序不使用标准的弹出控制部件定义函数，可以创建自己的控制部件定义函数，您可以选择使用 PopUpMenuSelect 函数来帮助您的应用程序处理弹出式菜单。这时，当光标在弹出式菜单中，用户按下鼠标按钮时，您的应用程序应该调用 PopUpMenuSelect 函数。您的应用程序必须在调用 PopUpMenuSelect 前醒目显示弹出标题，然后取消其醒目显示。PopUpMenuSelect 函数显示弹出式菜单，并在用户拖拽光标

通过菜单命令项目时，适当地醒目显示菜单命令项目。用户一释放鼠标按钮，PopUpMenuSelect 闪烁选择的项目（如果有话），并向您的应用程序返回表明所选项目的信息。您的应用程序负责醒目显示菜单标题或取消其醒目显示，更新弹出方框中的文字，并且当您使用 PopUpMenuSelect 函数时储存菜单命令项目设定状态的任何更改。

当您的应用程序需要为用户提供几种选择时，弹出式菜单很适用。注意，在用户选择显示弹出式菜单前，弹出式菜单对用户隐藏了这些选择。当用户不需要在所有的时间都看到所有的选择时，应使用弹出式菜单。例如，图 3-27 显示了一个对话框，它使用一个弹出式菜单，使用户能够在多种色彩的列表中选择一种色彩。

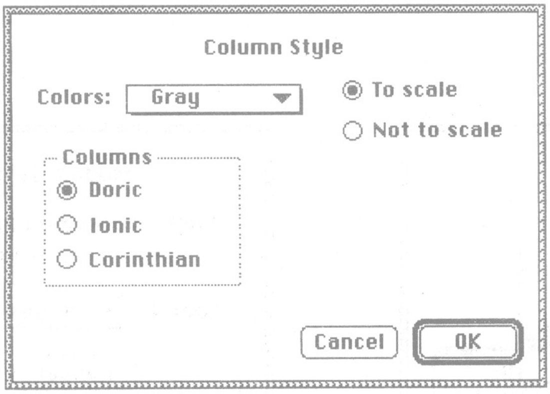


图 3-27　从多种属性的列表中选择一种属性

若您只需要显示几种选择，您可能会发现使用复选框和选项按钮对您的应用程序更方便。例如，在图 3-27 中，选择不是用弹出式菜单，而是用选项按钮列出的。只要可能，您应该为用户显示所有可用的选择。注意，在这个例子中，选项按钮占用的空间和一个相应的弹出式菜单差不多。

可以使用弹出式菜单让用户能够从一个多项选择组中获取一个选项。用户不能将弹出式菜单作为一种多重选择列表（一种以上的选定项）使用。若您这样做了，菜单方框中的文字将不能完全说明所执行的选项。例如，不能使用弹出式菜单进行字形的选择。在对话框中，字形的选择用复选框来执行更为恰当。图 3-28 显示了一个对话框，它让用户使用复选框，而不是弹出式菜单选择一种以上的字形。在这个实例中，大小和字体的选择用弹出式菜单进行，因为用户只能从列表的许多种选择中选定一种大小和一种字体。

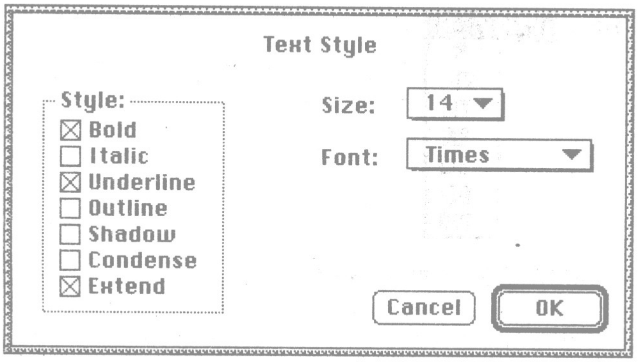


图 3-28　一个带有复选框和弹出式菜单的对话框

决不要使用弹出式菜单作为提供许多命令的方法。弹出式菜单不应该包含动作（动词），但可以包含属性（形容词）或允许用户从许多种选择中选一种设定状态。因此，您不应该在弹出式菜单中使用等效命令组合键。

如果合适的话，您的应用程序也可以使用键入的弹出式菜单。可使用一个键入的弹出式菜单，给用户一系列选项，并允许用户键入一个附加的选项。实现弹出式菜单的标准弹出控制部件定义函数对键入的弹出式菜单不提供特殊的支持。您可以创建自己的控制部件定义函数来处理键入的弹出式菜单。若您这样做，您的键入弹出式菜单应该遵循这里说明的准则。图 3-29 显示了关闭和打开状态下的一个典型的键入弹出式菜单。



图 3-29　一个典型的关闭和打开状态下的弹出式菜单

您的应用程序负责绘制并醒目显示弹出式菜单的键入域。您的应用程序不需要突出显示键入弹出式菜单的标题，而应该醒目显示键入域。

若用户键入菜单中已有的值，该项目成为当前项目。若用户键入的值和弹出式菜单中任何项目都不匹配，则将该项目放在菜单的顶部，并在它的下面添加一条分割线，将它和其它的标准项目分开。下一页的图 3-30 显示了一个添加有用户选项的键入弹出式菜单。



图 3-30　具有添加的用户选择的键入弹出式菜单

一个键入弹出式菜单应该允许用户键入一个添加的选项。即标准的键入弹出式菜单不在菜单中累积用户的选择。例如，若用户键入一个值 13，然后键入一个新的选项，比如 43，菜单将如图 3-30 所示，只是原来包含 13 的键入域和菜单命令项目现在被 43 代替了。

一个键入弹出式菜单还应该允许用户在菜单中键入标准值，或者在弹出式菜单中选择标准项目。若用户键入或选择标准项目，您应该删除原来添加在菜单上的用户指定的项目。例如，如图 3-30 所示，用户指定了一个非标准值 13。如果用户又键入或选择了 9，您的应用程序应该使弹出式菜单回到它原来如 3-27 页的图 3-29 所示的标准状态。

嵌套式菜单

嵌套式菜单是一个带有子菜单的菜单。当您的应用程序需要为用户提供附加的选项，而不占用菜单栏额外的空间时，嵌套式菜单是很有用的。若您在应用程序中使用嵌套式菜单，应用它来给用户提供附加选项或选择属性，而不是选择附加的命令。

在嵌套式菜单中，菜单命令项目将作为子菜单的标题；这个菜单命令项目包含一个表示该项目有一个子菜单的三角形。三角形出现在等效命令组合键的位置。子菜单的标题应该表示它所包含的选项。图 3-31 显示了一个菜单，它带有菜单标题为标签字形的子菜单。

当用户拖拽光标通过嵌套式菜单并停在一个带有子菜单的菜单命令项目上时，菜单管理器在一个短暂的延迟后显示子菜单。在用户浏览子菜单时，子菜单的标题仍然是醒目显示的；当用户释放鼠标按钮后，菜单管理器取消子菜单标题的醒目显示。

嵌套式菜单对提供有关项目的列表是非常有用的，例如字体大小和字形。决不要使用一个以上的嵌套式菜单（换句话说，不要给子菜单附属另一个子菜单）。您可以为一个子菜单的菜单命令项目指派等效命令组合键；但是这样会使用户难以迅速扫描所有菜单的等效命令组合键。

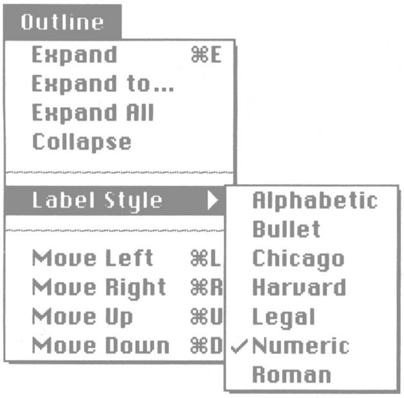


图 3-31 一个嵌套式菜单命令项目和它的子菜单

关于菜单管理器

菜单管理器和菜单定义过程及菜单栏定义过程一起，为您提供了在应用程序中方便地管理菜单的方法。菜单管理器使用两种数据结构：菜单记录和菜单列表。下两节介绍菜单管理器如何使用这两种数据结构。3-41 页的“使用菜单管理器”说明了在下列情况下您应如何使用菜单管理器：

- 使用 'MENU' 资源定义一个菜单
- 使用 'MBAR' 资源定义一个菜单栏
- 安装您的应用程序的菜单栏
- 更改菜单命令项目的外观
- 为菜单添加菜单命令项目
- 当用户选择一个菜单命令项目时，对用户做出响应
- 处理苹果菜单和帮助菜单
- 创建弹出式菜单
- 创建嵌套式菜单
- 当您的应用程序显示一个对话框时，处理对菜单的存取
- 编写您自己的菜单定义过程

菜单管理器如何保留有关菜单的信息

存储器管理器在菜单记录中保留有关菜单的信息。每一个菜单记录包括有关特定菜单的某些信息，包括：

- 菜单的菜单标识符
- 菜单的水平和垂直大小（以像素为单位）
- 菜单的菜单定义过程的一个间接指针
- 指示每个项目（前 31 个项目）是有效或无效，以及菜单标题是有效或无效的标志
- 菜单的内容，包括菜单标题和其它定义菜单命令项目的数据

通常您在一个菜单资源中指定上述的大多数信息。菜单资源是一个资源类型为 'MENU' 的资源。当您创建一个菜单时，菜单管理器将该信息储存在一个菜单资源中。菜单记录是一个 MenuInfo 类型的数据结构。您通常不需要直接存取菜单记录中的信息；当您对菜单进行任何更改时，例如添加一个菜单命令项目时，菜单管理器会自动更新菜单记录。若您需要直接存取菜单记录的域，请参阅 3-95 页的“菜单记录”。

菜单管理器通过被称为菜单标识符 的一个数字识别每个菜单。您必须在应用程序中给每个菜单指定一个菜单标识符。您的应用程序中的每个菜单必须有一个和该应用程序所有其它的菜单都不同的唯一的菜单标识符。您可以使用任何大于 0 的数作为一个下拉式菜单或弹出式菜单的菜单标识符；一个应用程序的子菜单只能使用 1 到 235 之间的数作为菜单标识符，桌面附件的子菜单必须使用 236 到 255 之间的数作为菜单标识符。当您创建一个菜单时，存储器管理器为该菜单创建一个菜单记录，并为该菜单记录返回一个间接指针。若要引用一个菜单，您通常应使用该菜单的菜单标识符 或者该菜单记录的间接指针。

若要引用一个菜单命令项目，应使用该菜单命令项目的项目编号。项目编号可以用来识别菜单中的项目；菜单中第一个项目的项目编号被指派为 1，第二个项目编号为 2，依次指派菜单中项目的项目编号，直至菜单中的最后一个菜单命令项目。

菜单管理器如何保留有关应用程序菜单栏的信息

菜单列表包含了一个或多个菜单的菜单记录的间接指针（虽然一个菜单列表也可能是空的）。菜单列表的结尾可以包含子菜单或弹出式菜单的菜单记录的间接指针；菜单列表的子菜单部分是指菜单列表中包含子菜单和弹出式菜单信息的部分。

当您的初始化应用程序菜单管理器时，菜单管理器指定当前菜单列表，它在一开始是空的。当前菜单列表的内容随着您的应用程序向菜单列表添加菜单或从菜单列表取消菜单而改变。

当前菜单列表包含当前菜单栏中所有菜单的菜单记录，以及您插入在当前菜单列表的子菜单和弹出式菜单的菜单记录。通常您的应用程序使用 GetNewMBar 来创建一个菜单列表，然后，它使用 SetMenuBar 将当前菜单列表设定到它新创建的菜单列表。您可以使用 GetMenu 函数和 InsertMenu 过程将其它菜单插入当前菜单列表中。

菜单管理器显示所有下拉式菜单的菜单栏和标题，它们是在您的应用程序调用 DrawMenuBar 过程时在当前菜单列表中定义的。菜单管理器按照在当前菜单列表中出现的次序显示菜单栏中的菜单。

菜单管理器为向当前菜单列表中添加菜单或从当前菜单列表中取消菜单提供了例程；您的应用程序决不应该直接存取一个菜单列表。若要引用一个菜单列表，应使用 GetNewMBar 或 GetMenuBar 返回的间接指针。

若您的应用程序调用 GetNewMBar 函数，而且您的菜单栏包括一个苹果菜单，菜单管理器将帮助菜单、键盘菜单（如果必要）及应用程序菜单插入您的应用程序的菜单列表；然后调用 SetMenuBar 将当前菜单列表设定到新创建的菜单列表中。若您的应用程序使用 InsertMenu 过程将苹果菜单插入当前菜单列表，菜单管理器还会将这些菜单插入应用程序的当前菜单列表。因此，您不应该对应用程序的当前菜单列表中的最后一个（或多个）菜单作任何假定。

当您的应用程序将一个子菜单插入当前菜单列表时，菜单管理器将该子菜单的菜单记录间接指针储存在当前菜单列表的子菜单部分。同样，当您的应用程序将一个弹出式菜单插入当前菜单列表时，菜单管理器将该弹出式菜单的菜单记录间接指针储存在当前菜单列表的子菜单部分。

使用菜单管理器

您可以将应用程序的菜单和菜单栏定义为资源，并使用菜单管理器例程来创建和管理它们。例如，用户在光标处于菜单栏时按下鼠标按钮，您的应用程序应该调用 MenuSelect 函数，允许用户从任何菜单选择一个命令。MenuSelect 函数处理用户的所有活动，直至用户释放鼠标按钮。当用户拖拽光标通过菜单栏时，MenuSelect 函数显示并取消菜单，当用户拖拽通过一个菜单时，它醒目显示有效的菜单命令项目。

您应该为应用程序的每个菜单标题和菜单命令项目提供旁述。您在资源中储存帮助旁述的信息和文字。有关如何为您的应用程序提供旁述的完全及专门的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh : More Macintosh Essentials*) 一书的“帮助管理器”一章。从 APDA 可以得到的 BalloonWriter 应用程序能够帮助您创建应用程序菜单的旁述。

在使用菜单管理器之前，您的应用程序需要初始化QuickDraw、字体管理器和窗口管理器。您的应用程序可以用 InitGraf、InitFont 和 InitWindows 过程来完成这个工作。若要初始化菜单管理器，应使用 InitMenus 过程。

若您的应用程序使用弹出式菜单，应使用 Gestalt 函数和 gestaltPopUpAttr 选项程序来确定弹出式菜单的控制部件定义函数是否可用。有关 Gestalt函数的信息，请参阅《Inside Macintosh 操作系统实用程序》(*Inside Macintosh : Operating System Utilities*) 一书。

若要在您 del 应用程序的菜单栏中创建下拉式菜单，您需要：

- 在 'MENU' 资源中创建每个下拉式菜单的描述
- 创建一个 'MBAR' 资源，它列出每个菜单的次序和资源标识符
- 使用 GetNewMBar 函数和 SetMenuBar 过程来设置您的菜单栏，并使用 DrawMenuBar 过程来绘制您的菜单栏

下一节“创建一个菜单”详细解释了这些步骤。

在创建了您的应用程序的菜单栏后，您可以启用或停用菜单命令项目、向菜单命令项目添加选择标记或破折号这类标记，或向您的菜单添加需要的项目。有关这些问题的信息，请参阅 3-58 页的“启用和停用菜单命令项目”，3-61 页的“更改菜单命令项目的标记”及 3-64 页的“为菜单添加命令项目”。

3-70 页的“处理菜单命令的用户选项”给出了在菜单栏中处理鼠标按下事件、调整应用程序的菜单、及确定用户是否选择了命令的等效命令组合键的方法。

3-78 页的“对用户选择一个菜单命令项目的响应”说明了当用户选择了一个项目后，您的应用程序应该如何响应，还给出了处理用户在苹果菜单和帮助菜单中选择一个命令的方法。

若希望您的应用程序显示对话框，请参阅 3-84 页的“从一个对话框存取菜单”。

最后，若您的应用程序需要创建子菜单或弹出式菜单，请参阅 3-53 页的“创建分级菜单”和 3-56 页的“创建弹出式菜单”。

创建菜单

您可以使用各种菜单管理器的例程来设置应用程序的菜单和菜单栏。您也可以使用所有这些方法来创建应用程序的弹出式菜单。

- 您可以在 'MENU' 资源中创建应用程序菜单的说明，并在 'MBAR' 资源中说明应用程序的菜单栏。您可以使用GetNewMBar函数来读取应用程序菜单栏和菜单的说明，并创建一个新的菜单列表；使用 SetMenuBar 过程将当前菜单列表设定到应用程序的菜单列表中；并使用 DrawMenuBar 过程来更新菜单栏
- 您可以在 'MENU' 资源中创建应用程序菜单的说明，用 GetMenu 将它们读入，用 InsertMenu 将它们插入当前菜单列表，并用 DrawMenuBar 更新菜单栏

■ 您可以使用 NewMenu 来创建新的空菜单；使用 AppendMenu、InsertMenuItem、InsertResMenu 或 AppendResMenu 在菜单中填入菜单命令项目；使用 InsertMenu 在当前菜单列表添加菜单；并使用 DrawMenuBar 来更新菜单栏

只要可能，您应该在菜单（'MENU'）资源中定义您的菜单，在菜单栏（'MBAR'）资源中定义您的菜单，这将使您的应用程序容易本地化。

若要创建一个嵌套式菜单，您需要创建子菜单和子菜单所属菜单的说明。您一般需要在'MENU'资源中创建这两种菜单的说明。通常您应使用 GetNewMBar 读入嵌套式菜单的说明（若您还提供了一个'MBAR'资源）。若要读入子菜单的资源并将它插入当前菜单列表，应使用 GetMenu 函数和 InsertMenu 过程。

若要创建一个弹出式菜单，应创建弹出式菜单和它的菜单命令项目的说明，创建一个使用推出控制部件定义函数的控制部件，并将该控制部件设置在一个窗口或对话框。您可以使用对话管理器或控制管理器的例程来显示及管理弹出式菜单。

菜单管理器为您的应用程序创建了一个菜单后，如果必要，您可以使用 AppendMenu、InsertMenuItem、InsertResMenu 或 AppendResMenu 将附加的菜单项目添加到菜单中。您可以使用各种菜单管理器的例程来改变菜单命令项目的外观。

下一节介绍了如何创建'MENU'和'MBAR'资源。3-53 页的“创建一个嵌套式菜单”介绍了如何创建一个具有子菜单的菜单，3-56 页的“创建一个弹出式菜单”介绍了如何创建弹出式菜单。

创建菜单资源

通常您应该在菜单（'MENU'）资源中定义您的菜单，以便实现菜单标题和菜单命令项目向其它语言、文化或地区的本地化。一个'MENU'资源定义了菜单的菜单标题和菜单中菜单命令项目的特征。列表 3-1 是一个应用程序苹果菜单的 Rez 格式的'MENU'资源示例。（Rez 是可从 MPW 得到的资源编译程序。您也可以用资源实用程序定义菜单，例如用 ResEdit，它可从 APDA 得到。）

程序清单 3-1　苹果菜单 'MENU' 资源的 Rez 输入

```
#define mApple 128

resource 'MENU' (mApple, preload) { /*resource ID, preload re-
source*/
    mApple,                  /*menu ID*/
    textMenuProc,            /*uses standard menu definition */
                             /* procedure*/
    0b1111111111111111111111111111101, /*enable About item, */
                                       /* disable divider, */
                                       /* enable all other items*/
```

```
    enabled,                        /*enable menu title*/
     apple,                         /*menu title*/
    {
                                    /*first menu item*/
        "About SurfWriter,          /*text of menu item */
                                    /* (includes ellipsis)*/
                                    /*item characteristics follow*/
            noicon,                 /*icon number (if any) or */
                                    /* script code (if any)*/
            nokey,                  /*keyboard equivalent (if any) */
                                    /* or submenu (if any) or */
                                    /* small or reduced icon (if any)*/
            nomark,                 /*marking character (if any) or */
                                    /* menu ID of submenu (if any)*/
            plain;                  /*style of menu item text*/
                                    /*second menu item*/
        "-",                        /*item text (divider)*/
            noicon, nokey, nomark, plain  /*item characteristics*/
        }
};
```

您在创建菜单时，还应该为应用程序的每个菜单命令项目和每个菜单标题定义帮助旁述。(3-31页的图 3-21显示了一个应用程序的“剪”命令的旁述。) 您在 'hmmu' 资源中定义应用程序菜单的旁述。有关如何创建‘hmmu’资源的实例，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Essentials*) 一书的“帮助管理器”一章。

程序清单 3-1 将苹果菜单的资源标识符定义为 128。您可以使用任何大于或等于128的数作为一个菜单的资源标识符。习惯上，许多应用程序使用 128 作为应用程序菜单栏中第一个菜单的资源标识符（苹果菜单)，并对以后的菜单资源标识符依次递增。

程序清单 3-1 还将苹果菜单的菜单标识符定义为 128。您的应用程序创建菜单后，菜单管理程序使用定义的菜单标识符来引用该菜单。您对一个菜单定义的菜单标识符数字并不一定要和资源标识符一样，但是使用同样的数字更方便些。您可以使用大于0的任何数作为下拉式菜单或弹出式菜单的菜单标识符; 应用程序的子菜单只能使用1到 235 之间的数作为菜单标识符; 桌面附件子菜单的菜单标识符必须在 236 到 255 之间。

该列表指定了这个菜单使用标准的菜单定义过程。若您希望创建您自己的菜单定义过程，应列出它的资源标识符，而不是 textMenuProc 常数。

在菜单定义过程的资源标识符之后是一个 32 位数（表示为 31 位域加上后面的一位布尔域），其中 1-31 位表明相应的菜单命令项目被启用或停用，第 0 位表明该菜单被启用或停用状态。

该列表在 31 位域中指定了第一个菜单命令项目应该是启用的，第二个菜单命令项目应该是停用的，以后的菜单命令项目（项目编号从 3-31）当菜单首次创建时应该是启用的。在创建了一个菜单后，您的应用程序可以使用 EnableItem 或 DisableItem 过程来启用或停用菜单命令项目。若一个菜单包含多于 31 个项目，菜单管理器在菜单启用时，自动启用第 31 个项目以后的所有项目。您的应用程序不能停用第 31 个项目以后的任何单个项目。但是您可以通过停用整个菜单来停用所有的项目，包括第 31 个项目以后的项目。

列表 3-1 指定了菜单标题在它首次创建时应该是启用的。您的应用程序可以使用 DisableItem 或 EnableItem 过程停用或启用菜单标题。当您使用 DisableItem 过程停用菜单时，菜单管理器停用菜单中的所有菜单命令项目（包括在第 31 个项目以后的所有项目），并使菜单标题色彩变淡。

资源列表使用常数 apple 标识菜单标题。若您指定 apple 常数作为标题，菜单管理器使用一个小苹果图标作为菜单标题。若显示器被设定为彩色显示，菜单管理器使用彩色苹果图标。然后该列表定义了菜单中每个菜单命令项目的特征。您需要对每个菜单命令项目定义菜单命令项目的文字和所有其它特征。例如，列表 3-1 将苹果菜单中的第一个项目定义为“关于...”命令；注意，该菜单命令项目的文字指定了三个省略号（...）。若您的应用程序显示一个对话框，在执行命令前需要从用户得到信息，应在菜单命令的文字后面指定 一个省略号。一般来说，若您的应用程序在用户选择一个菜单命令后，显示一个“确定警告”，您不应该使用省略号；“关于”命令是该准则的一个例外。

列表 3-1 定义了“关于”命令的其它特征—它在菜单命令项目文字的左边没有图标，没有等效命令组合键，在菜单命令项目文字的左边没有任何标记，以及菜单命令项目文字的字体字形是正常字形。

通过在图标域和等效命令组合键域指定值的各种组合，您可以定义一个图标（正常的、小的、缩小的和彩色的），一个等效命令组合键，一个子菜单或一个菜单命令项目的语系代码。注意，以下的段落说明了某些特征是互斥的（例如，一个项目可以有等效命令组合键或者子菜单，但不能二者兼有）。3-46页的表3-6总结了菜单管理器如何解释这些项目特征。

表 3-6　指定菜单资源中一个菜单命令项目的子菜单、语系代码、缩小图标、小图标和彩色图标

等效命令组合键域	图标域	标记字符域	说明
$1B		子菜单的菜单标识符	表明具有子菜单的项目。字符域指定了子菜单的菜单标识符。
$1C	项目文字的语系代码		
$1D	'ICON' 资源的图标号码		表明项目有一个用 'ICON' 资源定义的图标，它应该减小为16X16位矩形
$1E 的	'SICN' 资源的图标号码		表明项目有一个用 'SICN' 资源定义图标
$00 或 >$20	'ICON' 或 'cicn' 资源的图标号码		表明项目有一个用 'ICON' 或'cicn'资源定义的图标（在等效键盘域上大于$20的值指定了项目的等效命令组合键。）

若要给一个菜单命令项目指派一个图标，应在noicon常数的地方指定一个图标号码。您指定的图标号码应该是一个从1到255的数（对小图标或缩小图标，从1到254）；将您的图标号码加上256，将结果用于说明菜单命令项目图标的彩色图标（'cicn'）资源、图标（'ICON'）资源和小图标（'SICN'）资源的资源标识符。您必须在一个'cicn'、'ICON'或'SICN'资源中定义菜单命令项目的图标。菜单管理器只对您的菜单命令项目使用您定义的这些类型的图标。菜单管理器首先用计算的资源标识符搜索'cicn'资源，若找到了就使用该图标。若没有找到'cicn'资源（或者电脑没有彩色 QuickDraw），而且等效命令组合键域指定$1E，菜单管理器用计算的资源标识符搜索'SICN'资源。否则，除非等效命令组合键域包含$1D，菜单管理器将搜索'ICON'资源，并将它画在32 × 32位矩形中。若等效命令组合键域包含$1D，菜单管理器将缩小该图标，画在一个16 × 16位的矩形中。

若您提供一个 'ICON' 资源，并指定nokey常数或一个大于$20的值作为等效命令组合键，那么菜单管理器将会把整个菜单命令项目的矩形放大，来容纳32 × 32 位的 'ICON' 资源。若您指定 $1D 的值作为菜单命令项目等效命令组合键，菜单管理器将 'ICON' 资源缩小到16 × 16位矩形中。若您提供一个 'SICN' 资源，并指定$1E 的值作为菜单命令项目的等效命令组合键，菜单管理器在16 × 16位矩形中画小图标。若您提供一个 'cicn' 资源，菜单管理器自动根据 'cicn' 资源中指定的矩形放大菜单命令项目的矩形。（对于苹果菜单和应用程序菜单，菜单管理器可以自动缩小图标尺寸，从而使得图标能够恰好放入菜单命令项目的封闭矩形框内，或者使用应用程序图标库中的相应图标，如 'ics8' 资源，但要求该图标必须有效。）请参阅本书"Finder 界面"一章关于如何创建应用程序图标的详细说明。

如果要为一个菜单命令项目指派等效命令组合键,则应要求用户首先键入 Command 键,然后再键入一个您指定的字符,以取代该菜单命令项目资源定义中的 nokey 常数。如果您的应用程序希望在一个菜单命令项目上连接子菜单,那么指定 hierarchicalMenu 常数取代 nokey 常数即可。一个菜单命令项目可以有等效命令组合键或定义子菜单,但不能兼而有之。若要表示一个菜单命令项目拥有一个在'SICN'资源中定义的图标,可以指定 $ 1E 来取代 nokey 常数。若要表示一个菜单命令项目拥有一个在'ICON'资源中定义的图标,并表示菜单管理器应将该图标缩小成一个 16 × 16 位的矩形框,应指定 $ 1D 来取代 nokey 常数。带有小图标或缩小图标的菜单命令项目不能拥有等效命令组合键。

如果要设定菜单命令项目文字的语系代码,请指定 $ 1C 来取代 nokey 常数和定义希望的语系代码来取代 noicon 常数。如果一个项目在其等效命令组合键域中包含 $ 1C 并在其图标域中包含一个语系代码,那么菜单管理器就会用语系代码值指定的语系(如果安装了相应的语系系统的话)绘制项目文字。如果您没有指定菜单命令项目使用的语系代码,那么菜单管理器就会使用当前系统语系的系统字体来显示菜单命令项目文字。对于罗马语系,系统字体是 Chicago 系统字体大小是 12。

如果要指派菜单命令项目文字左边和图标左边的标志,请指定标志符号来取代资源定义中的 nomark 常数。如果菜单命令项目有一个子菜单,则可以指定子菜单的菜单标识符来取代 nomark 常数。应用程序的子菜单必须使用 1 ~ 235 的菜单标识符。桌面附件的子菜单必须使用 236 ~ 255 的菜单标识符。请注意,在'MENU'资源中定义子菜单的菜单标识符不能将子菜单挂接在它的菜单上。您必须使用 GetMenu 函数和 InsertMenu 过程才能完成这项工作。从 3 ~ 53 页的"创建嵌套式菜单"中有关于挂接菜单的子菜单的说明。

如果要指派菜单命令项目的字形,请在您的'MENU'资源中使用常数 bold、italic、plain、outline 和 shadow 以便获取其相应的字形。

程序清单 3-1 将第二个菜单命令项目定义成了分隔线。如果您使用一个破折号作为定义菜单命令项目文字字符串中的第一个字符时,菜单管理器将创建一个长度与整个菜单命令项目宽度相等的分隔线。您不能为分隔线指派任何其它特征。

Apple 菜单的'MENU'资源不列出其它任何菜单命令项目。在您的应用程序创建了菜单之后,可以使用 AppendResMenu 过程将桌面项目添加到苹果菜单中。请参阅 3 - 68 页上"为 Apple 菜单添加项目"中的详细说明。

只要您创建了一个菜单,您就可以使用 AppendMenu、InsertMenuItem、InsertResMenu 或 AppendResMenu 过程将附加项目附加到该菜单中。您也可以使用菜单管理器的例程来更改各个菜单命令项目的特征。请参阅 3 - 57 页的"更改菜单中的项目外观"中的详细说明。

3 – 24 页上的图 3 – 12 是一个典型的应用程序编辑菜单。列表 3 – 2 是该编辑菜单的 'MENU' 资源。

程序清单 3 – 2 一个编辑菜单 'MENU' 资源的 Rez 输入

```
#define mEdit 130
resource 'MENU' (mEdit, preload) {/*resource ID, preload resource*/
   mEdit,                         /*menu ID*/
   textMenuProc,                  /*uses standard menu definition */
                                  /* procedure*/
   0b0000000000000000001001000000000,/*enable/disable first 31 menu */
                                  /* items as appropriate*/
     enabled,                     /*enable title*/
     "Edit",                      /*text of menu title*/
      {                           /*menu items*/
   "Undo",noicon, "Z",nomark,plain; /*keyboard equivalent Command-Z*/
   "-",   noicon, nokey,nomark, plain;
   "Cut", noicon,"X",nomark, plain; /*keyboard equivalent Command-X*/
   "Copy",noicon,"C",nomark,plain; /*keyboard equivalent Command-C*/
   "Paste",noicon,"V",nomark,plain; /*keyboard equivalent Command-V*/
   "Clear",noicon, nokey, nomark, plain;
   "Select All",
          noicon,"A",nomark, plain;/*keyboard equivalent Command-A*/
   "-",   noicon, nokey,nomark, plain;
   "Create Publisher...,
          noicon, nokey, nomark, plain;
   "Subscribe To...,
          noicon, nokey,nomark, plain;
   "Publisher Options...,
          noicon, nokey,nomark, plain;
   "-",   noicon, nokey,nomark, plain;
   "Show Clipboard",
          noicon, nokey,nomark, plain
     }
};
```

程序清单 3-2 将此编辑菜单的资源标识符定义为 130，菜单标识符定义为 130，并且为菜单指定了需使用的标准菜单定义过程。本列表定义了前 31 个菜单命令项目的初始有效状态，而且还指定首次创建菜单标题时应启用它。

资源列表将该菜单的标题定义为“编辑”。然后定义菜单中每个菜单命令项目的特征。对于每个菜单命令项目，您需要指定菜单命令项目的文字和菜单命令项目的其它特征。例如，列表 3 – 2 将编辑菜单中的第一个项目定义为“取消”命令，其特征是：菜单命令项目文字的左边没有图标，有一个 Command-Z 等效命令组合键，该菜单命令项目文字的左边没有任何标志，而且菜单命令项目文字使用的是正常字形。该列表将第二个菜单命令项目定义为分隔线。定义了“剪”、“复制”和“贴”命令；指定了每个命令的等效命令组合键；而且定义了菜单中的其它项目。

程序清单 3 – 3 是另外一个实例即一个典型的应用程序文件菜单的资源说明。

程序清单 3 – 3 一个文件菜单 'MENU' 资源的 Rez 输入

```
resource 'MENU' (mFile, preload) {
     mFile, textMenuProc,
     0b0000000000000000000010000000000,
     enabled,
     "File",
     {
          "New",         noicon, "N",       nomark, plain;
          "Open...,      noicon, "O",       nomark, plain;
          "-",           noicon, nokey,     nomark, plain;
          "Close",       noicon, "W",       nomark, plain;
          "Save",        noicon, "S",       nomark, plain;
          "Save As...,   noicon, nokey,     nomark, plain;
          "-",           noicon, nokey,     nomark, plain;
          "Page Setup...,noicon, nokey,     nomark, plain;
          "Print...,     noicon, "P",       nomark, plain;
          "-",           noicon, nokey,     nomark, plain;
          "Quit",        noicon, "Q",       nomark, plain
          }
};
```

创建菜单栏资源

您通常需要使用菜单栏（'MBAR'）资源来定义应用程序的菜单栏。程序清单 3 – 4 是一个示例应用程序的 Rez 格式的 'MBAR' 资源。

程序清单 3 – 4 'MBAR' 资源的 Rez 输入

```
#define rMenuBar            128

#define mApple              128
#define mFile               129
#define mEdit               130
```

```
resource 'MBAR' (rMenuBar, preload) {/*resource ID, preload*/
     /*menus appear in the order listed here*/
     { mApple, mFile, mEdit };        /*resource IDs for menus in */
                                      /* this menu bar*/
};
```

程序清单 3 - 4 定义'MBAR'资源的资源标识符是 128。这个'MBAR'资源定义了其中菜单的顺序和资源标识符；它将前 3 个菜单的资源标识符分别定义为 128、129 和 130。菜单管理器在从'MBAR'资源中创建菜单栏时通常使用指派的资源标识符来读取菜单。

设置应用程序的菜单栏

如果要创建一个'MBAR'资源中的定义菜单列表，可以使用 GetNewMBar 函数。对于'MBAR'资源定义的每个菜单，GetNewMBar 函数都将为它们创建一个菜单记录、并根据其对应的'MENU'资源中的资源定义创建每个菜单，而且还将每个菜单插入新的菜单列表。然后，GetNewMBar 函数返回一个已创建菜单列表的间接指针。例如，此代码段可以为资源标识符 为是 128 的'MBAR'资源定义的菜单栏创建一个菜单列表（由常数 rMenuBar 定义）。

```
CONST
     rMenuBar = 128;
VAR
   menuBar:          Handle;
 menuBar := GetNewMBar(rMenuBar);{read menus and menu bar }
                                 { descriptions,create & return }
                                 { a handle to a new menu list}
```

请使用SetMenuBar过程将您的应用程序创建的菜单列表设定为当前菜单列表，使用 DrawMenuBar 过程更改菜单栏的外观。例如，程序清单 3-5 就是使用这两个例程设定应用程序菜单栏的。

程序清单 3 - 5 设定一个应用程序的菜单和菜单栏

```
PROCEDURE MyMakeMenus;
VAR
     menuBar:                Handle;
 BEGIN
{first use the GetNewMBar function to read menus in & create a }
{ new menu list. If you define an Apple menu, the Menu Manager }
{ inserts the Help and Application menus (and Keyboard menu if }
{ necessary) into the newly created menu list}
```

```
    menuBar := GetNewMBar(rMenuBar);
    IF menuBar = NIL THEN
         EXIT(MyMakeMenus);
    SetMenuBar(menuBar);{insert menus into the current menu list}
    DisposHandle(menuBar);
                        {add desktop items in Apple Menu Items }
                        { folder to Apple menu}
    AppendResMenu(GetMenuHandle(mApple), 'DRVR');
    MyAdjustMenus;      {adjust items and enabled state of menus}
    DrawMenuBar;        {draw the menu bar}
END;
```

程序清单 3 - 5 中的代码段是通过读资源标识符 为 128 的'MBAR'资源中的定义来创建应用程序菜单栏的，并使用 SetMenuBar 将新创建的菜单列表设定成了当前菜单列表。然后，该代码段使用 AppendResMenu 过程将 Apple 菜单命令项目文件夹中的桌面项目添加到了 Apple 菜单中。

您可以使用 GetMenuHandle 函数为当前菜单列表中的任何菜单的菜单记录获取一个间接指针。如果您将选中的菜单之菜单标识符 作为一个参数提供给 GetMenuHandle, etMenuHandle 就会向该菜单的菜单记录返回一个间接指针。多数菜单管理器的例程都需要用菜单标识符或一个间接指针作为菜单记录的参数。

在创建了菜单栏并添加了其它必要的菜单或项目之后，该代码就会调用 MyAdjustMenus 过程来调整应用程序的菜单—例如，此过程可以根据应用程序的当前状态设定菜单命令项目的有效和无效状态。(3 - 74 页上的程序清单 3 - 19 就是程序清单 3 - 5 中使用的应用程序定义的 MyAdjustMenus 过程。) 调整好菜单后，程序清单 3 - 5 中的代码段便使用 DrawMenuBar 按照菜单当前的有效状态和当前菜单列表中的菜单定义在菜单栏中绘制菜单。

通常，您会使用'MENU'资源和一个'MBAR'资源来定义应用程序的菜单及其菜单栏，并使用 GetNewMBar 函数读这些资源定义。您也可以选择使用 GetMenu 函数读'MENU'资源或者使用 NewMenu 创建新的空菜单。您可以使用 InsertMenu 过程将一个菜单插入当前菜单列表。请参阅 3 - 105 页上的“创建菜单”和 3 - 108 页上的“在当前菜单列表中添加和删除菜单”中关于使用这些例程组建菜单的说明。

如果您的应用程序需要使用子菜单，您就应该使用 GetMenu 函数和 InsertMenu 过程将这些菜单用于您的应用程序。请参阅 3 - 53 页“创建嵌套式菜单”中关于创建子菜单的说明。如果您的应用程序需要使用弹出式菜单，您可以使用弹出控制定义功能函数和对话管理器或控制管理器的例行程序来创建和显示弹出式菜单。请参阅 3 - 56 页的“创建弹出式菜单”中关于创建弹出式菜单的说明。

当您的应用程序调用 GetNewMBar 时，菜单管理器将创建和初始化您的应用程序的菜单色彩信息表。如果您希望在菜单和菜单栏中使用系统设定色彩之外的其它色彩，您可以在您的应用程序的菜单色彩信息表中添加输入。通过提供菜单色彩信息表（'mctb'）资源或使用 SetMcEntries 过程，您就可以将输入添加到此表中。通常您应使用系统设定色彩，因为这样有助于保持一致的用户介面。

如果您要将彩色输入添加到应用程序的菜单色彩信息表中，而且您的应用程序需要使用一个以上的菜单栏，那么在更改菜单栏之前，您就需要存储一份应用程序菜单色彩信息表。在调用 GetNewMBar 之前使用 GetMCInfo 函数，然后调用 SetMCInfo 还原菜单色彩信息表。程序清单 3 － 6 所示的例程可以存储菜单色彩信息表并可在创建新的菜单栏时还原它。

程序清单 3 － 6 存储和还原菜单色彩信息

```
PROCEDURE MyChangeMenuBarAndSaveColorInfo;
CONST
     rMenuBar2 = 129;
VAR
   menu:                  MenuHandle;
   menuBar:               Handle;
   currentMCTable:        MCTableHandle;
   newMCTable:            MCTableHandle;
 BEGIN
   currentMCTable := GetMCInfo;  {save menu color info table}
   IF currentMCTable = NIL THEN
     EXIT(MyChangeMenuBarAndSaveColorInfo);
   menuBar:=GetNewMBar(rMenuBar2);{read menus in & create new menu list}
   IF menuBar = NIL THEN
     EXIT(MyChangeMenuBarAndSaveColorInfo);
   newMCTable := GetMCInfo;      {get new menu color info table}
   IF newMCTable = NIL THEN
     EXIT(MyChangeMenuBarAndSaveColorInfo);
   SetMCInfo(currentMCTable);    {restore previous menu color info table}
   SetMenuBar(menuBar);          {insert menus into the current menu list}
   DisposHandle(menuBar);
   AppendResMenu(GetMenuHandle(m2Apple),'DRVR');{add desktop items from }
                                 { Apple Menu Items folder to Apple menu}
   MyAdjustMenus;                {adjust menu items}
   DrawMenuBar;                  {draw the menu bar}
 END;
```

创建嵌套式菜单

在一个或多个菜单命令项目上挂接有一个子菜单的那种菜单称为嵌套式菜单。子菜单有助于您的应用程序向用户提供附加选项，且不必占用菜单栏的额外空间。如果您在应用程序中使用了子菜单，您就可以用它来为用户提供附加选项或属性选项，而不是附加命令。

下拉式菜单的菜单命令项目是它所挂接的子菜单的标题。在菜单命令项目的等效命令组合键位置上有一个指向右方的三角形，表示该菜单命令项目挂接了一个子菜单。子菜单的标题应体现它所包含的选项特点。图 3－32 是一个带有子菜单的菜单，子菜单的标题是 Label Style（标签字形）。

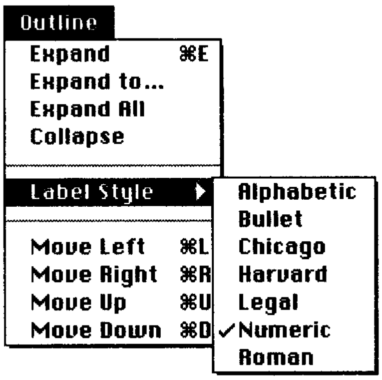


图3-32　带有子菜单的菜单命令项目

当用户将光标拖拽过一个菜单并停留在一个挂接有子菜单的菜单命令项目上时，菜单管理器会在暂短的延迟后显示该子菜单。当用户浏览该子菜单时，子菜单的标题保持醒目显示状态；当用户释放鼠标按钮时，菜单管理器便不再醒目显示子菜单的菜单标题。

您的应用程序负责在子菜单选项或属性旁边设置标志。例如，图 3－32 中，应用程序在 Numeric 子菜单命令项目旁设置了“✓”标志符号表示它是当前选项。如果用户在该菜单中选择了新的选项，您的应用程序应相应地更新菜单命令项目。

当您在‘MENU’资源中定义菜单命令项目时，可以通过标志其特征（使用 hierarchicalMenu 常数）来指定某特定菜单命令项目挂接一个子菜单。您不能对带有子菜单的菜单命令项目指派等效命令组合键。(您可以为子菜单的菜单命令项目定义等效命令组合键，但不建议这样做。）您可以使用子菜单的菜单标识符 取代菜单命令项目资源描述中的标志符号。因此，带有子菜单的菜单命令项目不能有标志符号而且不能有等效命令组合键。

如果希望将一个子菜单插入当前菜单列表，您必须使用 InsertMenu 过程。GetNewMBar 函数不能读取任何子菜单的资源描述。

程序清单 3 - 7 是一个称为 Outline(空心字形)的应用程序定义菜单的'MENU'资源 Outline 菜单中包含若干个菜单命令项目，其中有 Label Style（标签字形）菜单命令项目。这个菜单命令项目的说明中包含常数 hierarchicalMenu，表示该项目带有一个子菜单。另外这个菜单命令项目说明中还包含子菜单的菜单标识符（由常数 mSubMenu 定义)。应用程序子菜单的菜单标识符 必须是从 1 至 235；桌面附件子菜单的菜单标识符必须是从 236 至 255。

定义该子菜单的菜单带有 Label Style 菜单命令项目指定的菜单标识符。定义子菜单菜单命令项目的方法与定义下拉式菜单之菜单命令项目的方法相同(只是您不应在子菜单中定义等效命令组合键，而且不应将一个子菜单挂接到另一个子菜单上)。

程序清单 3-7　带有子菜单的嵌套式菜单的 Rez 输入

```
#define mOutline 135
#define mSubMenu 181

resource 'MENU' (mOutline, preload) {
        mOutline ,                              /*menu ID*/
        textMenuProc,
        0b0000000000000000000000000010000,
            enabled,
        "Outline",                              /*menu title*/
        {                                       /*menu items*/
            "Expand",      noicon, "E",             nomark, plain;
            "Expand To...,noicon, nokey,            nomark, plain;
            "Expand All", noicon, nokey,            nomark, plain;
            "Collapse",    noicon, nokey,           nomark, plain;
            "-",           noicon, nokey,           nomark, plain;
            /*the Label Style item has a submenu with menu ID mSubMenu*/
            "Label Style",noicon, hierarchicalMenu, mSubMenu, plain;
            "-",           noicon, nokey,           nomark, plain;
            "Move Left",   noicon, "L",             nomark, plain;
            "Move Right", noicon, "R",              nomark, plain;
            "Move Up",     noicon, "U",             nomark, plain;
            "Move Down",   noicon, "D",             nomark, plain
        }
};
```

```
resource 'MENU' (mSubMenu , preload) {
    mSubMenu ,                          /*menu ID*/
        textMenuProc,
        0b0000000000000000000000001111111,
            enabled,
        "Label Style",                  /*menu title (ignored--defined */
                                        /* by parent menu item text)*/
        {                               /*menu items*/
            "Alphabetic",       noicon, nokey, nomark, plain;
            "Bullet",           noicon, nokey, nomark, plain;
            "Chicago",          noicon, nokey, nomark, plain;
            "Harvard",          noicon, nokey, nomark, plain;
            "Legal",            noicon, nokey, nomark, plain;
            "Numeric",          noicon, nokey, nomark, plain;
            "Roman",            noicon, nokey, nomark, plain
    }
};
```

在您使用 GetNewMBar 读菜单说明并创建一个新菜单列表时，GetNewMBar 将在菜单记录中记录子菜单的菜单标识符，但不读取子菜单的说明。若要读一个子菜单的说明，可以使用 GetMenu 函数。若要将一个子菜单真正插入到当前菜单列表中，您必须使用 InsertMenu 过程。

需要时，您的应用程序可以使用 GetMenu 函数从 'MENU' 资源中读一个菜单的特征说明。GetMenu 函数能够为菜单创建菜单记录，并在您的应用程序堆中分配菜单的记录空间。GetMenu 函数将根据菜单的 'MENU' 资源创建菜单和菜单命令项目（并填写菜单记录）。 GetMenu 函数不能将菜单插入菜单列表。如果您希望将它添加到当前菜单列表，可以使用 InsertMenu 过程。

程序清单 3－8 使用 GetMenu 函数读子菜单说明并使用 InsertMenu 过程将菜单插入当前菜单列表。

程序清单 3－8 创建嵌套式菜单

```
PROCEDURE MyMakeSubMenu (subMenuResID: Integer);
VAR
    subMenu:            MenuHandle;
BEGIN
    subMenu := GetMenu(subMenuResID);
    InsertMenu(subMenu, -1);
END;
```

如果您希望使用 InsertMenu 过程将子菜单插入当前菜单列表，那么就请将第二个参数指定为－1，然后将该菜单插入菜单列表的子菜单分区中即可。当用户在菜单中上下移动选择菜单命令项目时，如果一个菜单命令项目带有一个子菜单，MenuSelect 函数便会在菜单列表的子菜单分区中查找该子菜单；它要搜索一个带有定义菜单标识符 的菜单，而这个定义菜单标识符是与嵌套式菜单命令项目指定的菜单标识符相匹配的。如果找到了带有匹配菜单标识符的菜单，它就会将子菜单挂接到该菜单命令项目上，让用户浏览该子菜单。

创建弹出式菜单

在 System 7 中，弹出式菜单是作为控制部件来实现的。您可以使用定义其它菜单（使用 'MENU' 资源）的方法定义弹出式菜单的菜单命令项目，通过使用标准的弹出式控制部件定义函数的控制，您可以定义弹出式菜单自身的一些特殊功能（如弹出式菜单的位置）。弹出式菜单为用户提供了一种不必移到菜单栏就可以在选择清单中进行选择的简单途径。这种菜单尤其适用于需要用户指定多种设定状态或值的对话框。图 3－33 是一个对话框中的弹出式菜单实例。

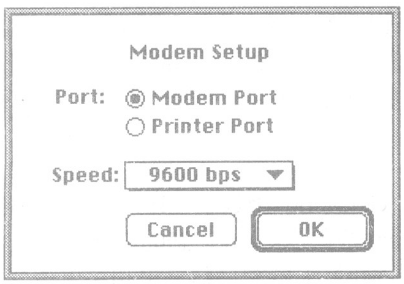


图 3-33 对话框中的弹出式菜单

如果要创建一个弹出式菜单，应首先创建一个使用弹出式控制部件定义函数的控制器，定义弹出式菜单及其菜单命令项目，并将该控制器与一个窗口或对话框联系起来。您可以使用对话管理器或控制管理器例程显示弹出式菜单。

例如，如果您定义了一个包含有弹出控制器的定模式对话框并使用对话管理器显示和帮助处理对话框中的事件，则对话管理器会自动地利用弹出式控制部件定义函数来绘制该控制器，并在用户将光标置于一个弹出式控制器上按下鼠标按钮后处理与用户的交互作用。

如果您的应用程序在一个窗口中定义了控制器，您可以使用 TrackControl 和其它控制管理器例程来处理弹出式菜单。

弹出式控制部件定义函数可以在弹出框周围绘制一个方框、绘制阴影(DropShadow)、将文字插入弹出框、绘制指向下方的三角形，以及绘制弹出标题。如果一个对话框中含有使用弹出式控制部件定义函数的控制器，而且用户在光标位于弹出式控制器时按下鼠标按钮，那么弹出式控制部件定义函数就会醒目显示弹出式菜单标题、显示该弹出式菜单、并处理与用户之间的所有交互作用，直至用户释放鼠标按钮。用户释放鼠标按钮，弹出式控制部件定义函数便关闭弹出框、绘制用户在弹出框中作出的选择（或者，如果用户没有进行新的选择，则还原先前的选项）、将用户的选择作为控制值存储起来，并且不再醒目显示弹出式菜单标题。您的应用程序可以使用控制管理器的函数 GetControlValue 获取控制值和确定弹出式菜单中的当前选择项目。

若要创建一个弹出式控制器，则应创建一个控制器并通过指定 popupMenuProc 常数来指定该控制器使用弹出式控制部件定义函数：

```
CONST popupMenuProc=1008 ;   {pop-up menu control}
```

如果您将 popupMenuProc（加上相应的变量代码）指定为一个控制器的资源描述 procID 域，那么当您的应用程序创建控制器时（使用对话管理器或使用 GetNewControl），控制管理器就会创建一个弹出式控制器，其中包括弹出标题和带有 1 个像素阴影的弹出框。弹出标题的外观和菜单中的值由资源（或传送到 NewControl 的其它参数）中储存的其它值控制。请参阅本书“控制管理器”一章中关于 NewControl 函数的说明。

如果您的应用程序没有使用标准的弹出式控制部件定义函数，您可以创建您自己的控制部件定义函数以实现弹出式菜单。对于这种情况，您可以使用 PopUpMenuSelect 函数来绘制弹出式菜单和跟踪该菜单内的光标。在调用 PopUpMenuSelect 之前和之后，您的应用程序负责醒目显示弹出式菜单标题和取消其醒目显示（以便重复菜单栏中菜单标题的动作）。如果需要使用 PopUpMenuSelect 函数，您的应用程序还必须在弹出式菜单中设定相应的项目标志。

请参阅本书的“控制管理器”一章中关于创建控制器的详细说明。若想了解定义图 3-33 所示的对话框列表的有关信息，请参阅本书的“对话框管理器”一章。

更改菜单中的项目外观

您可以使用菜单管理器例程更改菜单命令项目的外观。例如，您可以更改字形、文字、或菜单命令项目的其它特征。您也可以启用或停用一个菜单命令项目。

多数菜单管理器例程都可以获取或设定特定菜单命令项目的特征，它们要求下面三个参数：

- 一个菜单记录间接指针，要求该菜单含有理想的菜单命令项目
- 菜单命令项目编号
- 一个变量，用于指定要设定的数据或标识返回项目信息的位置

启用和停用菜单命令项目

您可以使用 EnableItem 和 DisableItem 过程来启用和停用指定的菜单命令项目或整个菜单。要实现这一操作，您必须将以下两个参数传送给这两个过程，一个是间接指针，用于标识理想菜单的菜单记录，另一个是项目编号，用于标识启用或停用的特定菜单命令项目，或者是一个 0 值，它表示整个菜单应该被启用或停用。

在调用 MenuSelect 或 MenuKey 之前，您的应用程序应始终根据用户的意愿适当地启用或停用菜单命令项目。例如，当剪贴数据存储区包含有用户可贴的数据时，您就应该启用 P“贴”命令。（在 3－74 页的程序清单 3－19 中列示了调整应用程序菜单的代码。）

在您停用或启用整个菜单时，应调用 DrawMenuBar 以更新菜单栏。DrawMenuBar 过程可以根据菜单当前的有效状态以及当前菜单列表中菜单的定义来绘制菜单栏中的菜单。

如果停用了整个菜单，菜单管理器就会在您的应用程序下次调用 DrawMenuBar 时淡化显示该菜单标题，并且在显示该菜单时淡化显示其所有的项目。如果启用了整个菜单，菜单管理器则只是启用菜单标题和您先前没有单独停用的项目；如果您的应用程序曾使用该菜单命令项目的项目编号调用 DisableItem 而停用了某些项目，则菜单管理器不启用它们。例如，如果应用程序的“编辑”菜单中的所有项目均为有效项目，您可以使用 DisableItem 单独停用“剪”和“复制”命令。如果您将 0 作为菜单命令项目参数传送到 DisableItem 以选择停用整个菜单，则该菜单及其所有的项目便被停用。然后，如果您将 0 作为菜单命令项目参数传送到 EnableItem 以启用整个菜单，则该菜单及其项目被启用，但剪切和拷贝命令仍保持无效状态。在这种情况下，若要启用剪和复制命令，您必须使用 EnableItem 单独启用每个命令。

您可以使用 DisableItem 来停用那些有时不太适当的项目。例如，当用户未选择任何数据进行剪或贴时，您可以停用“剪”和“复制”命令，当剪贴数据为空存则可以停用“贴”命令。

此代码段可以启用文件菜单、停用编辑菜单中的剪和复制命令，以及停用应用程序定义的菜单色彩。

```
VAR
    menu : MenuHandle ;

    menu  := GetMenuHandle (mFile) ;   {get a handle to the File menu}
    EnableItem (menu,  0) ;            {enable File menu and any items }
                                           { not individually disabled}
    DrawMenuBar ;                          {update menu bar's appearance}

    menu  := GetMenuHandle (mEdit) ;   {get a handle to the Edit menu}
    DisableItem (menu,  iCut) ;        {disable the Cut command}
    DisableItem (menu,  iCopy) ;       {disable the Copy command}
```

```
menu  := GetMenuHandle (mColors) ; {get a handle to Colors menu}
DisableItem (menu,  0) ;           {disable Colors menu & all }
                                       { items in it}
DrawMenuBar ;                          {update menu bar's appearance}
```

如果您停用或启用了整个菜单，当您需要更新菜单栏的外观时，可以调用 DrawMenuBar。如果您不需要立即更新菜单栏，您也可以使用 InvalMenuBar 过程，而不用 DrawMenuBar，从而减少了菜单栏的闪烁。不像前面的实例那样绘制两次菜单栏，您可以使用 InvalMenuBar 来代替 DrawMenuBar，使用事件管理器在下一次扫描更新事件时重新绘制菜单栏。InvalMenuBar 过程可以用于 System 7 和更新系统。请参阅 3－114 页中关于 InvalMenuBar 过程的详细说明。

更改项目文字

您可以使用菜单管理器例程来获取或设定菜单命令项目的文字。

若要获取一个菜单命令项目的文字，可以使用 GetMenuItemText 过程。例如，您可以使用 GetMenuItemText 过程来获取您曾利用 InsertResMenu 或 AppendResMenu 添加到菜单中的菜单命令项目文字。

若要设定一个菜单命令项目的文字，可以使用 SetMenuItemText 过程。您可以使用一种简便的设定方法，即使用 SetMenuItemText 过程来更改允许用户在两种状态之间转换的菜单命令文字。例如，如果您的应用程序，有一个允许用户显示或隐藏剪贴板窗口的菜单命令，根据该窗口是否为当前显示，您就可以使用 SetMenuItemText 过程在适当时候更改该菜单命令项目的文字。

在程序清单 3－9 中，显示的是根据应用程序定义（存有状态信息）的全局变量 (gToggLestate) 的状态，将菜单命令项目的文字从隐藏剪贴板更改为显示剪贴板，或反之。

程序清单 3－9 更改菜单命令项目的文字

```
PROCEDURE MyToggleHideShow ;
VAR
    myMenu :            MenuHandle ;
    item :          Integer ;
    itemString :        Str255 ;
BEGIN
    myMenu  := GetMenuHandle (mEdit) ;
    item  := iToggleHideShow ;
    IF gToggleState = kShow THEN
    BEGIN
        GetIndString (itemString,  kMyStrings,  kShowClipboard) ;
        gToggleState  := kHide ;
    END ;
```

```
    ELSE
    BEGIN
        GetIndString (itemString, kMyStrings, kHideClipboard) ;
        gToggleState := kShow ;
    END ;
    SetMenuItemText (myMenu, item, itemString) ;
END ;
```

请注意，如果使用 SetMenuItemText 过程，您就应该在一个字符串资源或字符串列表资源中定义菜单命令项目的文字（例如，使用一个`STR' 或`STR#' 资源)。这样可以使您的应用程序易于本地化。

更改菜单命令项目的字形

您可以使用 SetItemStyle 或 GetItemStyle 过程来更改或获取菜单命令项目的字形。若要设定一个菜单命令项目的字形，则需为菜单记录指定一个间接指针，该菜单中包含有您希望设定的菜单命令项目，指定要设定的菜单命令项目的编号，以及指定理想的字形。

您可以使用字形数据类型定义的集的值来指定字形：

```
TYPE
    StyleItem = (bold, italic, underline, outline, shadow,
                condense, extend) ;
    Style = SET OF StyleItem ;
```

您可以将一个菜单命令项目的字形设定为 0. 1，或设定成一个以上的 StyleItem 数据类型定义的字形。您也可以将一个菜单命令项目的字形设定到空集以获取正常字型。

程序清单 3 — 10 是设定菜单命令项目（列在一个应用程序的字形菜单中）字形的代码。

程序清单 3 - 10　　设定菜单命令项目的字形

```
VAR
    menu :       MenuHandle ;
    itemStyle : Style ;
    menu  := GetMenuHandle (mStyle) ;   {get a handle to the Style menu}
    itemStyle  :=  [italic] ;
    SetItemStyle (menu, iItalic, itemStyle) ; {set to italic style}
    itemStyle  :=  [bold] ;
    SetItemStyle (menu, iBold, itemStyle) ; {set item to bold style}
    itemStyle  :=  [bold, Italic] ;
    SetItemStyle (menu, iBoldItal, itemStyle) ; {bold & italic style}
    itemStyle  :=  [] ;
    SetItemStyle (menu, iPlain, itemStyle) ; {set item to plain style}
```

您可以使用 GetItemStyle 过程来获取一个菜单命令项目的字形。

更改菜单命令项目的标记

您可以使用 SetItemMark 或 GetItemMark 来改变或获取菜单命令项目的标记。如果要把菜单命令项目的标记设定为检查标记,您可以使用 CheckItem 或 SetItemMark 过程。

如果想设定一个菜单命令项目的标记,则应为菜单记录指定一个间接指针,您要设定标记的菜单命令项目包含在该菜单中,另外还需要指定设定菜单命令项目的编号,以及指定作为菜单命令项目标记字符的标记。

一般情况下,您可以在菜单中使用选择标记和破折号,菜单中包含有设定属性的命令,您应该将这些命令分成多个相容命令项目组。例如,您可以在字形菜单中使用选择标记和破折号的组合来表示选择项是否包含一种以上的字形。3 - 15 页的图 3 - 18 就是一个在菜单中使用选择标记和破折号的例子。从3 - 14 页的“菜单命令项目分组”给出了为菜单命令项目分组的准则。

您可以通过给 SetItemMark 过程传递一个字符作为它的一个参数来指定菜单命令项目的标记。在您的菜单命令项目中,只应使用标准的标记字符,如检查标记、菱形图案或破折号;避免使用其它可能使用户混淆的标记。您可以使用这里列出的常数来指定菜单命令项目没有标记,或将选择标记(✓)和菱形图案(◆)指定为标记符号。

```
CONST    noMark        = 0;       {no marking character}
         checkMark     = $12;     {checkmark}
         diamondMark   = $13;     {diamond symbol}
```

程序清单 3 - 11 是在菜单中使用标记的另一个例子,程序清单 3 - 11 给出了在一个应用程序定义的目录菜单中设定项目标记的代码段。它将被存取的倒数第一个目录的菜单命令项目标记符号设定为选择标记,将被存取的倒数第二个目录的标记符号设定为菱形标记,并删除了被存取的倒数第三个目录的标记。

程序清单 3 - 11 添加菜单命令项目标记和删除菜单命令项目标记

```
VAR
    menu:          MenuHandle;
    itemMark:      Char;
                                 {get handle to Directory menu}
    menu := GetMenuHandle(mDirectory);

    itemMark := CHR(checkMark);
    SetItemMark(menu, gLastDir, itemMark);{set mark to checkmark}

    itemMark := CHR(diamondMark);
    SetItemMark(menu, gOldLastDir, itemMark);{set mark to diamond}

    itemMark := CHR(noMark);
    SetItemMark(menu, gSecondLastDir, itemMark);{remove any mark}
```

您也可以使用 CheckItem 过程把一个菜单命令项目的标记设定为选择标记。

```
VAR
    menu:          MenuHandle;
                                {get handle to Directory menu}
    menu := GetMenuHandle(mDirectory);
    CheckItem(menu, gLastDir, TRUE);      {set to checkmark}
    CheckItem(menu, gSecondLastDir, FALSE);{remove checkmark or }
                                          { any other mark}
```

更改菜单命令项目的图标或语系输入代码

您可以使用 SetItemIcon 或 GetItemIcon 过程来改变或获取菜单命令项目的图标。您还可以使用这些过程来获取或设定菜单命令项目文字的语系代码。

如果想使用 SetItemIcon 过程来设定菜单命令项目的语系代码，您需要:

- 为菜单记录指定一个间接指针，该菜单记录所对应的菜单中包含着您要为其设定语系代码的项目
- 指定需要设定代码的菜单命令项目的编号
- 指定语系代码

若要设定一个菜单命令项目的语系代码，您还必须把项目的等效命令组合键域定义为 $1C。如果一个项目在其等效命令组合键域中包含 $1C，在其图标域中包含语系代码，并安装了相应的语系系统，菜单管理器将会按照语系代码值所确立的语系绘制该项目。

若想使用 SetItemIcon 过程设定菜单命令项目的图标，您需要:

- 为菜单记录指定一个间接指针，该菜单记录所对应的菜单中包含着您要为其设定图标的项目
- 指定需要设定代码的菜单命令项目的编号
- 指定图标编号（菜单管理器使用图标编号来生成图标的资源标识符）

对于彩色图标或图标，您指定给 SetItemIcon 的图标编号必须是一个从 1 到 255 的值；对于小图标或缩小图标，您指定的编号必须是一个从 1 至 254 的值。如果值为 0 则表示该项目没有图标。菜单管理器会在您指定的编号上再加 256，然后使用计算所得的号码作为图标的资源标识符。例如，如果您指定的图标号码为 5，菜单管理器则使用资源管理器来查找资源标识符 为 261 的图标。菜单管理器首先寻找一个'cicn'类型的图标资源，如果用计算所得的资源标识符号码不能找到一个图标资源(或电脑没有 Color QuickDraw)，而且等效命令组合键域包含 $1E 时，它将寻找一个类型为 'SICN' 的资源；否则，寻找 'ICON' 资源。

如果您只想提供一个黑白图标，则可以使用 'ICON' 或 'SICN' 资源。另外，如果装有 Color QuickDraw，而且您希望菜单管理器使用彩色图标，那么就可以提供一个 'cicn' 资源。图 3 - 34 是一个菜单命令项目的几个图标的实例，这些图标是根据图标资源而生成的：一个是 'SICN' 资源，一个是 'ICON' 资源，一个是缩小了的 'ICON' 资源，相当于一个 16 × 16 位的矩形，另外一个 'cicn' 资源。



图 3-34 菜单命令项目中的图标

菜单管理器可以按照您的规格要求自动调整菜单命令项目矩形的尺寸或图标的大小，以便图标能够恰如其分地装入菜单命令项目。如果菜单管理器使用的是一个 'cicn' 资源，它将根据 'cicn' 资源中指定的矩形自动放大菜单命令项目的封闭矩形。如果菜单管理器使用的是一个 'ICON' 资源并且项目将 nokey 常数指定为等效命令组合键，那么菜单管理器将放大菜单命令项目矩形以适应 32 × 32 位的 'ICON' 资源。您也可以通过将一个 ＄1D 值指定为项目的等效命令组合键来请求菜单管理器将一个 'ICON' 资源缩小为 16 × 16 位大小的小图标。如果要请求菜单管理器使用一个 'SICN' 资源而不是一个 'ICON' 资源，那么就应该将一个 ＄1E 值指定为项目的等效命令组合键。

本代码段将一个菜单命令项目的图标设定成了一个指定图标。

```
VAR
    menu:          MenuHandle;
    itemIcon:      Byte;

    itemIcon := 5;
    menu := GetMenuHandle(mWeather);
    {set the icon for this item in the Weather menu}
    SetItemIcon(menu, iBeachWeather, itemIcon);
```

程序清单 3 - 12 是 3 个菜单命令项目的 Rez 描述，每个项目都包含着若干个图标。第一个菜单命令项目有一个资源标识符为 261 (5 加 256) 的图标，既可以由资源类型 'cicn' 定义也可由 'ICON' 来定义。第二个菜单命令项目有一个资源标识符 为 262 (6 加 256) 的图标，既可以由一个 'cicn' 资源确定也可由一个 'ICON' 资源确定。但是，在这种情况下，这个 ＄1D 值会请求菜单管理器将 'ICON' 资源缩小为一个小图标。第三个菜单命令项目中有一个资源标识符为 263 (7 加 256) 的图标，也是既可以由一个 'cicn' 资源定义又可以让一个 'SICN' 资源来定义。

程序清单 3 － 12 指定菜单命令项目图标

```
#define mWeather 138
resource 'MENU' (mWeather, preload) {
        mWeather,
        textMenuProc,
```

```
        0b00000000000000000001011101100111,
        enabled, "Weather",
        {
             "Beach Weather",     /*item has icon or color icon */
                                  /* with icon number 5*/
                  5, nokey, nomark, plain;
             "Ski Weather",       /*item has reduced icon or color */
                                  /* icon with icon number 6*/
                  6, $1D, nomark, plain;
             "Kite-Flying Weather",    /*item has small icon or */
                                   /* color icon with icon number 7*/
                  7, $1E, nomark, plain
        }
   };
```

请参阅本书的“Finder 界面”一章中关于怎样创建图标的详细介绍。

添加菜单命令项目

通常，您都是在一个‘MENU’资源中定义菜单及其项目的。有时是在创建了菜单之后，给菜单再添加一些命令项目。当您使用 NewMenu, GetMenu 或 GetNewMBar 创建一个菜单之后，您可以使用 AppendMenu, InsertMenuItem, AppendResMenu 或 InsertResMenu 过程来为这个菜单添加命令项目。

您可以使用 AppendResMenu 或 InsertResMenu 为一个菜单添加由资源名称组成的项目。例如您可以使用 AppendResMenu 过程为应用程序的字体菜单添加字体，或从苹果菜单命令项目文件夹中将所有桌面环境项目加到应用程序的苹果菜单中。当需要把‘MENU’资源中没有定义的项目加到菜单中时，这些就是常见的例子。请参阅 3－69 页的“添加菜单字体”和 3－68 的“苹果菜单添加命令项目”中关于给菜单添加资源名称的介绍。

如果您需要给应用程序的帮助菜单添加项目，您就必须使用 AppendMenu 或 InsertMenu 来添加这些额外的项目。本节讨论怎样使用 AppendMenu 和 InsertMenuItem 过程来添加项目，并且在 3－67 页的“给帮助菜单添加命令项目”部分给出了一个为帮助菜单添加命令项目的具体实例。

如果您需要给一个预先创建的菜单增加除资源名称以外的项目，可以使用 AppendMenu 或 InsertMenuItem 过程。您可以使用 AppendMenu 在一个菜单的末尾增加项目；注意，当使用 AppendMenu 时，您只能在菜单的末尾增加项目。而使用 InsertMenuItem 可以在菜单中任何给定的项目之后增加项目。当您使用 AppendMenu 或 InsertMenuItem 给一菜单增加项目时，您可以为菜单命令项目指定与定义‘MENU’资源时使用的特性一样的特性。

对于您希望在其中增加一个或多个项目的菜单,您应该为它的菜单记录指定一个间接指针。然后,指定一个说明项目的字符串,将这个字符串作为一个参数传给 AppendMenu 或 InsertMenuItem 过程。您传递给这些过程的字符串应该包括文字以及菜单命令项目的任何特性。您可以将连字符指定为菜单命令项目文字以创建一个菜单命令项目分隔线。您还可以在文字字符串中使用各种元字符来分隔菜单命令项目并指定菜单命令项目的某些特性。元字符在菜单中不显示出来。

下面是一个您可以在文字字符串中使用的元字符列表，这里的字符串是为 AppendMenu 或 InsertMenuItem 过程指定的。

元字符	说明
; 或回车	分隔菜单命令项目。
^	当后面接一个图标编号时，可以用来定义项目图标。如果等效命令组合键域中含有 $1C，那么这个号码就可以被解释为一个语系代码。
!	后面接一个字符时，可以定义项目的标记。如果等效命令组合键域中含有 $1B，那么该值就可以解释为这个菜单项的一个子菜单的菜单标识符。
<	当后面接着 B、I、U、O 和 S 中的一个或多个字符时，则分别将该项目的字符的字形定义为黑体、斜体、下画线、空心或阴影。
/	后面接一字符时，可以定义该项目的等效命令组合键。当后面接 $1B时，则可以指定该菜单命令项目有一个子菜单。如果希望为菜单命令项目指定一个语系代码、一个小图标或缩小图标，可以使用 SetItemCmd 过程分别把等效命令组合键域设定为 $1C、$1D 或 $1E。
(	定义该菜单命令项目无效。

您可以在文字字符串利用这些元字符,以便定义一个菜单命令项目的特性。注意您指定的元字符不在菜单命令项目中显示。(如果需要在一个菜单命令项目的文字中使用一个元字符，则首先应使用 AppendMenu 或 InsertMenuItem 至少指定一个字符作为项目的文字。然后，使用 SetMenuItemText 过程把项目文字设定为理想的字符串。)

附 注

如果使用 AppendMenu 或 InsertMenuItem 过程来添加菜单命令项目，那么您就应该在资源中定义文字和标记或等效命令组合键以便更容易地定位。

程序清单 3-13 是一字符串列表（'STR#'）资源，它储存了下个实例中使用的菜单命令项目的文字。

程序清单 3 － 13　　菜单命令项目文字的 Rez 输入

```
resource 'STR#' (300, "Text for appended menu items") {
    {
    /*[1]*/
    "Just Text";
    /*[2]*/
    "Pick a Color";
    /*[3]*/
    "(^2!=Everything<B/E";
    }
);
```

这段代码使用 AppendMenu 过程来添加一个菜单命令项目，该菜单命令项目除了文字是由myMenu变量中的菜单间接指针标识以外，再没有其它的特性。该菜单命令项目的文字是“Just Text”，储存在资源标识符 为 300 的'STR＃'资源中。

```
VAR
    myMenu:        MenuHandle;
    itemString:    Str255;

    myMenu := GetMenuHandle(mLibrary);
    GetIndString(itemString, 300, 1);
    AppendMenu(myMenu, itemString);
```

如果希望在给定的菜单命令项目之后插入一个项目，可以使用 InsertMenuItem 过程。例如上面这段代码就是将菜单命令项目“选取一种色彩”插入了用 iRed 常数指定项目编号的菜单命令项目之后。新菜单命令项目的文字包括字符串“选取一种色彩…”，储存在资源标识符 为 300 的'SIR＃'资源中。

```
VAR
    myMenu:        MenuHandle;
    itemString:    Str255;

    myMenu := GetMenuHandle(mColors);
    GetIndString(itemString, 300, 2);
    InsertMenuItem(myMenu, itemString, iRed);
```

如果您在传递给 AppendMenu 或 InsertMenuItem 的文字字符串中没明确地为一项目特性指定一个值，那么该过程就会为这种特性分配系统缺省设定值。菜单管理器定义的系统设定项目特性有这样几种：“没有图标”，“没有标记符号”，“没有等效命令组合键”和“正常字形”。

如果您没有作特别指定，AppendMenu 或 InsertMenuItem 通常把所添加的菜单命令项目置为有效状态。

这段代码给使用变量 myMenu 中的菜单间接指针标识的菜单添加了一个文字为“Everything”的菜单命令项目。该菜单命令项目的文字和其它特性储存在程序清单 3－13 显示的'STR#'资源中。而且还指定：这个菜单命令项目无效，有一个资源标识符为 258（2＋256）的图标，并且标记符号为“=”；文字的字形为黑体；并且菜单命令项目的等效命令组合键为 Command-E。

```
VAR
    myMenu:        MenuHandle;
    itemString:    Str255;

    myMenu := GetMenuHandle(mLibrary);
    GetIndString(itemString, 300, 3);
    AppendMenu(myMenu, itemString);
```

这段代码使用 AppendMenu 为编辑菜单附加了多个项目。

```
    myMenu:        MenuHandle;

    myMenu := GetMenuHandle(mEdit);
    AppendMenu(myMenu, 'Undo/Z;-;Cut/X;Copy/C;Paste/V');
```

当文字字符串指定了多个命令项目时，在怎样处理给定的字符串方面 InsertMenuItem 过程与 AppendMenu 不同。InsertMenuIterm 过程以文字字符串中相反的顺序插入菜单命令项目。例如，这段代码可以使用 InsertMenuItem 在编辑菜单中插入菜单命令项目，其结果与前面的例子相同。

```
VAR
    myMenu:        MenuHandle;

    myMenu := GetMenuHandle(mEdit);
    InsertMenuItem(myMenu, 'Paste/V';Copy/C;Cut/X;-;Undo/Z',0);
```

一旦您给一个菜单添加了一个菜单命令项目，您就可以使用菜单管理器例程更改这个项目的一切特性，在 3－57 页“更改一个菜单中项目的外观”中有这方面的说明。

在帮助菜单中添加项目

您可以使用 HMGetHelpMenuHandle 函数和 AppendMenu 或 InsertMenuItem 过程其中之一给帮助菜单添加项目。

HMGetHelpMenuHandle函数可以返回一个应用程序的帮助菜单的菜单记录间接指针拷贝。请不要使用 GetMenuHandle 函数来获取帮助菜单的间接指针，因为 GetMenuHandle 返回一个全局帮助菜单的间接指针，而不是您的应用程序的帮助菜单间接指针。一旦您拥有了您的应用程序特有的帮助菜单间接指针，您就可以使用 AppendMenu 过程或其它菜单管理器例程给帮助菜单添加项目。例如，程序清单 3－14 添加 3－30 页图 3－19 中显示的菜单命令项目。

程序清单 3－14 给帮助菜单添加项目

```
PROCEDURE MyAddHelpItem;
VAR
    myMenu:        MenuHandle;
    myErr:         OSErr;
    itemString:    Str255;
BEGIN
    myErr := HMGetHelpMenuHandle(myMenu);
    IF myErr = noErr THEN
        IF myMenu <> NIL THEN
        BEGIN
            {get the string (with index kSurfHelp) from the 'STR#' }
            { resource with resource ID kMyStrings}
            GetIndString(itemString, kMyStrings, kSurfHelp);
            AppendMenu(myMenu, itemString);
        END;
END;
```

在您给帮助菜单添加项目时，帮助管理器将会自动地在标准帮助菜单命令项目的结尾和新添项目之间添加一个菜单命令项目分隔线。

一定要使用 `hmnu' 资源并指定 kHMHelpMenuID 常数作为资源标识符，以便为添加到帮助菜单中的项目提供旁述。(帮助管理器能够自动为帮助菜单标题和标准的帮助菜单命令项目创建旁述。) 请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh : More Macintosh Toolbox*) 一书的“帮助管理器”一章中关于 `hmnu' 资源的专门介绍。

当用户在帮助菜单中选择标准菜单命令项目时，帮助管理器将自动处理该事件。当用户从帮助菜单中选择“显示旁述”或“隐藏旁述”时，帮助管理器会自动启用或停用“帮助”功能。对“旁述帮助”的设定是全局性的，会影响到所有的应用程序。请参阅 3－81 页“处理帮助菜单”关于用户选择新添项目时您应如何作出响应的介绍。

在苹果菜单中添加项目

如果希望将包含在苹果菜单命令项目文件夹中的项目插入应用程序的苹果菜单，需要使用 AppendResMenu 或 InsertResMenu 过程并指定 `DRVR' 作为资源类型。这样做可以使这个过程自动地将苹果菜单命令项目文件夹中的所有项目添加到指定的菜单中。

任何桌面对象用户都可以将它们放置在苹果菜单命令项目文件类中。当用户在这个文件夹中放入一项目时，系统软件会自动地将它添加到所有已打开的应用程序的苹果菜单中的项目列表。

在您的应用程序的菜单栏中插入苹果菜单之后(通过使用 GetNewMBar 或 GetMenu 和 InsertMenu 过程),您的应用程序就可以给它添加项目了。程序清单 3 – 15 是一个使用 GetMenuHandle 来获取应用程序苹果菜单间接指针的代码段。这个代码段使用 AppendResMenu 过程,规定 AppendResMenu 应该将苹果菜单命令项目文件夹中的所有桌面项目添加到应用程序的苹果菜单中。

程序清单 3 – 15　　给苹果菜单增加菜单命令项目

```
VAR
    myMenu:                              MenuHandle;

    myMenu := GetMenuHandle(mApple);
    IF myMenu <> NIL THEN
        AppendResMenu(myMenu, 'DRVR'); {add desktop items in the }
                                       { Apple Menu Items folder }
                                       { to the Apple menu}
```

3 – 70 页中的程序清单 3 – 16 是一个完整示例:设置了应用程序的菜单栏,给苹果菜单添加项目,给字体菜单添加项目,然后更新菜单栏的外观。

添加字体菜单

如果您的应用程序提供了一个字体菜单,那么您一般地应在一个 'MENU' 资源中包括有 关于该菜单的说明,在一个 'MBAR' 资源中包括有关于菜单栏的说明,并使用 GetNewMBar 来创建您的菜单栏和菜单栏中的所有菜单。一旦创建了菜单,您就可以使用 AppendResMenu 在系统文件夹的字体文件夹中添加所有字体资源的名称(或在比 7.1 版本更早的系统软件中,或在系统文件中),然后用这些字体资源名称作为您的应用程序的字体菜单中的菜单命令项目。(您还可以使用 InsertResMenu 将字体插入您的菜单。)

程序清单 3 – 16 将会告诉您怎样向一个应用程序的字体菜单添加字体文件夹中的字体资源名称。AppendResMenu 过程将指定类型的所有资源添加到一个给定菜单中。如果您指定了资源类型 'FOND' 或 'FONT',菜单管理器就会将所有 'FOND' 和 'FONT' 类型的资源添加到菜单中。('NFNT' 和 'sfnt' 资源是通过 'FOND' 资源来指定的。)

AppendResMenu 和 InsertResMenu 过程可以对那些字体编号超过 $4000 的字体资源执行特别处理。菜单管理器将会自动地把菜单命令项目的等效命令组合键域设定为 $1C,并在菜单命令项目的图标域为这种 'FOND' 资源储存语系代码。如果安装了某种字体的语系系统,菜单管理器将以其相应的语系显示该字体名称。

程序清单 3 － 16 为菜单添加字体名称

```
VAR
    menu:             MenuHandle;
    menuBar:          Handle;
 BEGIN
                      {read menus in & create new menu list}
    menuBar := GetNewMBar(rMenuBar);
         IF menuBar = NIL THEN
       EXIT(MyMakeAllMenus);
    SetMenuBar(menuBar);{insert menus into the current menu list}
    DisposHandle(menuBar);
    myMenu := GetMenuHandle(mApple);
    IF myMenu <> NIL THEN            {add desktop items in }
       AppendResMenu(myMenu,'DRVR'); { Apple Menu Items }
                                     { folder to Apple menu}
    myMenu := GetMenuHandle(mFont);
    IF myMenu <> NIL THEN
       AppendResMenu(myMenu, 'FONT');{add font names to the }
         { Font menu--this adds all bitmapped and TrueType fonts }
         { in the Fonts folder to the Font menu}
    MyAddHelpItem;          {add app-specific item to Help menu}
    MyAdjustMenus;          {adjust menu items}
    DrawMenuBar;            {draw the menu bar}
 END;
```

您的应用程序应该在列出字体名称的菜单命令项目文字之前设置正确的标记，以便向用户指明当前字体。(3 － 61 页“更改菜单命令项目的标记”解释了怎样将标记添加至菜单命令项目或从菜单命令项目中删除标记；3 － 26 页图 3 － 13 和 3 － 27 页图 3 － 14 显示了典型的字体菜单实例。)

如果您的应用程序允许用户改变文字的字形或字体大小，您就应该提供分离的大小和字形菜单。请参阅 3 － 82 页的“处理大小菜单”中关于在您的应用程序中提供一个“字体大小菜单”的介绍。

处理用户选择的菜单命令

如果当光标处在菜单栏中时用户按下鼠标按钮，您的应用程序应该首先调整它的菜单（启用菜单命令项目或停用菜单命令项目，并为与用户的上下文相适应地采取行动，将标记符号添加至各项目或从项目中删除标记符号），然后调用 MenuSelect 函数以便用户选择一个菜单命令。MenuSelect 函数将处理所有用户对话，直到用户释放鼠标按钮并返回一个值作为它的函数结果为止，这个值可以表示用户选择了哪个项目。

对于一个带有等效命令组合键的命令,您的应用程序应该允许用户通过按下与该菜单命令的等效命令组合键相对应的键来选择该命令。如果用户按下 Command 键和另一键,您的应用程序就应该调整它的菜单并随后调用 Menukey 函数把这种组合映射到一个等效命令组合键。Menukey 函数返回一个值作为它的功能结果,指出与等效命令组合键相对应的菜单和菜单命令项目。

当用户选择了一个菜单命令,您的应用程序就应该执行与该命令相关联的动作。MenuSelect 和 Menukey 函数醒目显示含有被选菜单命令的菜单标题。在您的应用程序执行完毕与用户所选的菜单命令相关联的所有操作之后,您的应用程序应该使用 HiliteMenu 过程取消菜单标题的醒目显示。

但是,如果响应菜单命令时您的应用程序显示一个含有可编辑文字的窗口(如定模式对话框),您应该立即取消菜单标题的醒目显示,以便用户存取编辑菜单或其它相应的菜单。换句话说,无论用户何时使用菜单,都要确保该菜单标题处于非醒目显示状态。

当用户选择了一个菜单命令,而这个命令涉及一个耗时很长的操作时,则应显示动画手表光标或显示一个执行状态对话框,告诉用户操作正在进行。

如果您需要让用户记录他们的动作(例如菜单命令、文字输入或任何动作序列)以便将来重放,那么每当用户执行一个重要动作时,您的应用程序就应该给它自己发送苹果事件。如果要对菜单命令作这样的处理,您的应用程序一般都是给它自己发送一个苹果事件以执行与所选的苹果命令相关联的动作。例如,当用户从文件菜单中选择了 New 命令时,您的应用程序就可以给它自己选择发送一个 Create Element 事件。然后,您的应用程序创建新文稿以响应该事件。如果想了解发送苹果事件以响应苹果菜单命令的有关信息,请参阅《Inside Macintosh 应用程序间通信》(*Inside Macintosh: Interapplication Communication*) 一书。

下面几节介绍了您的应用程序怎样才能完成如下任务:

- 当光标处在菜单栏中时判断用户是否按下了鼠标按钮
- 在显示菜单之前或在响应用户选择的一个命令的等效组合键之前,调整其菜单——根据文稿的当前状态启用或停用相应的命令
- 确定用户是否选择了一个菜单命令的等效组合键
- 当用户选择一个菜单命令时做出响应

下面几节还讨论了当用户从苹果或帮助菜单中选择一个项目时,您的应用程序应该如何响应的问题。

处理菜单栏中的鼠标按下事件

当光标处在菜单栏时，您可以通过检验事件记录中是否有一个鼠标按下事件来确定用户是何时何地按下鼠标按钮的。您可以使用窗口管理器函数 FindWindow 将鼠标按下事件发生时鼠标的位置映射到屏幕上相应的区域。如果光标处在菜单栏中，您的应用程序应该调用 MenuSelect 函数，允许用户选择一个菜单命令。

程序清单 3 - 17 一个应用程序定义的过程 DoEvent，它可以用来确定是否有一个鼠标按下事件发生，如果有，还将调用另一个应用程序定义的过程来处理鼠标按下事件。(如果想了解关于怎样处理事件的全面讨论，请参阅本书"事件管理器"一章。)

程序清单 3 - 17　确定一个鼠标按下事件是否发生

```
PROCEDURE DoEvent (event: EventRecord);
BEGIN
     CASE event.what OF
          mouseDown:              {handle mouse-down event}
               DoMouseDown(event);
          {handle other events appropriately}
     END; {of CASE}
END;
```

程序清单3-18是一个应用程序定义的例程DoMouseDown，它可以处理鼠标按下事件。DoMouseDown 过程可以确定鼠标按钮按下时光标的位置并作出正确的响应。

程序清单 3 - 18　判断光标何时在菜单栏中

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
     part:          Integer;
     thisWindow:    WindowPtr;
BEGIN
     part := FindWindow(event.where, thisWindow);
     CASE part OF
          inMenuBar:{mouse down in menu bar, respond appropriately}
               BEGIN
                    {adjust marks and enabled state of menu items}
                    MyAdjustMenus;
                    {let user choose a menu command if desired}
                    DoMenuCommand(MenuSelect(event.where));
               END;
```

```
        {handle other mouse-down events appropriately}
    END; {of CASE}
END;
```

您可以使用 FindWindow 函数将用户按下鼠标按钮时鼠标的位置映射到屏幕的通用区域。如果鼠标位置处在菜单栏中，FindWindow 函数将返回常数 inMenuBar 。在程序清单 3 - 18 中，如果与鼠标按下事件相关联的鼠标位置处在菜单栏中，DoMouseDown 过程首先调用另一个由应用程序定义的过程 MyAdjustMenus 来调整菜单显示状态。程序清单 3 - 19 显示了 MyAdjustMenus 过程。

然后，DoMouseDown 过程调用一个应用程序定义的过程 DoMenuCommand。DoMouseDown 过程把从 MenuSelect 函数返回的值作为一个参数传递给 DoMenuCommand 过程。

MenuSelect 函数显示菜单并处理所有用户对话，直到用户释放鼠标按钮为止。MenuSelect 函数返回一个长整数，表示用户是否选择了一个菜单命令，如果选择了，它还指示用户选择的究竟是哪个菜单和哪个命令。

3 - 79 页程序清单 3 - 24 显示的是 DoMenuCommand 过程。

调整应用程序菜单

您的应用程序应该总是在调用 MenuSelect 或 MenuKey 之前调整它的菜单。例如，您应该根据需要启用菜单命令项目或停用菜单命令项目，并将选择标记(✓)或破折号添加到项目中作为项目的属性。当您调整应用程序菜单时，您应该根据当前的窗口类型启用菜单命令项目或停用菜单命令项目。例如，当一个文稿窗口处在最前面时，您应该启用那些适合文稿窗口的项目。当一个不定模式对话框或定模式对话框在最前面时，则应使用那些适合这种特殊对话框的项目。程序清单 3 - 19 是一个应用程序定义的例程 MyAdjustMenus，它能够正确调整应用程序 Surfwriter 的菜单。

MyAdjustMenu 首先确定当前窗口的类型，然后正确地调整应用程序的菜单。应用程序定义的过程 MyGetWindowType 返回一个值，表明窗口是一个文稿窗口，还是一个对话窗口，或属于一桌面附件的窗口。如果没有当前窗口，则返回常数 kNil。(请参阅本书“窗口管理器”一章关于 MyGetWindowType 过程的程序清单。) 另外，MyAdjustMenus 过程还可以调用其它由应用程序定义的例程来调整菜单以适应给定的窗口类型。

程序清单 3 － 19　　调整一个应用程序菜单

```
PROCEDURE MyAdjustMenus;
VAR
    window:             WindowPtr;
    windowType:         Integer;
BEGIN
    window := FrontWindow;
    windowType := MyGetWindowType(window);
    CASE windowType OF
    kMyDocWindow:
    BEGIN {document window is in front, adjust items appropriately}
        MyAdjustFileMenuForDocWindow;
        MyAdjustEditMenuForDocWindow;
        {adjust other menus as needed}
    END;     {of adjusting menus for a document window}
    kMyDialogWindow:
        {adjust menus accordingly for any dialog box}
        MyAdjustMenusForDialogs;
    kDAWindow:{adjust menus accordingly for a DA window}
        MyAdjustMenusForDA;
    kNil:{adjust menus accordingly when there isn't a front window}
        MyAdjustMenusNoWindows;
    END; {of CASE}
    DrawMenuBar;
END;
```

程序清单 3－20 是一个应用程序定义的过程myAdjustFileMenuForDocWindow。这个过程可以根据文稿的状态，针对应用程序的文稿窗口启用和停用文件菜单中的命令项目。例如，这个应用程序始终允许用户创建一个新文稿或打开一个文件，因而该代码段可以启用 “创建新文稿”和“打开”菜单命令项目。另外，该代码段还可以启用 “关闭”、“存储为”、“页面设置”、“打印”和“退出”菜单命令项目生效。如果用户上次存储文件后又修改了文件，那么该代码段将会启用存储命令；否则，停用“存储”命令处于无效状态。

程序清单 3 － 20　　为一个文稿窗口调整文件菜单

```
PROCEDURE MyAdjustFileMenuForDocWindow;
VAR
    window:                 WindowPtr;
    menu:                   MenuHandle;
    myData:                 MyDocRecHnd;
```

```
BEGIN
    window := FrontWindow;
    menu := GetMenuHandle(mFile);    {get a handle to the File menu}
    IF menu = NIL THEN               {add your own error handling}
        EXIT (MyAdjustFileMenuForDocWindow);
    EnableItem(menu, iNew);
    EnableItem(menu, iOpen);
    EnableItem(menu, iClose);
    myData := MyDocRecHnd(GetWRefCon(window));
    IF myData^^.windowDirty THEN
        EnableItem(menu, iSave)
    ELSE
        DisableItem(menu, iSave);
    EnableItem(menu, iSaveAs);
    EnableItem(menu, iPageSetup);
    EnableItem(menu, iPrint);
    EnableItem(menu, iQuit);
END;
```

程序清单 3－21 是一个应用程序定义的过程 MyAdjustEditMenuForDocWindow。

程序清单 3－21　为一个文稿窗口调整编辑菜单

```
PROCEDURE MyAdjustEditMenuForDocWindow;
VAR
    window:               WindowPtr;
    menu:                 MenuHandle;
    selection, undo:      Boolean;
    isSubscriber:         Boolean;
    undoText:             Str255;
    offset:               LongInt;
BEGIN
    window := FrontWindow;
    menu := GetMenuHandle(mEdit);    {get a handle to the Edit menu}
    IF menu = NIL THEN               {add your own error handling}
        EXIT (MyAdjustEditMenuForDocWindow);
    undo := MyIsLastActionUndoable(undoText);
    IF undo THEN        {if action can be undone}
    BEGIN
        SetMenuItemText(menu, iUndo, undoText);
        EnableItem(menu, iUndo);
    END
```

```
    ELSE                {if action can't be undone}
    BEGIN
        SetMenuItemText(menu, iUndo, gCantUndo);
        DisableItem(menu, iUndo);
    END;
    selection := MySelection(window);
    IF selection THEN
    BEGIN    {enable editing items if there's a selection}
        EnableItem(menu, iCut);
        EnableItem(menu, iCopy);
        EnableItem(menu, iCreatePublisher);
    END
    ELSE
    BEGIN    {disable editing items if there isn't a selection}
        DisableItem(menu, iCut);
        DisableItem(menu, iCopy);
        DisableItem(menu, iCreatePublisher);
    END;
    IF GetScrap(NIL, 'TEXT', offset) > 0 THEN
        EnableItem(menu, iPaste) {enable if something to paste}
    ELSE
        DisableItem(menu, iPaste);{disable if nothing to paste}
    EnableItem(menu, iSelectAll);
    EnableItem(menu, iSubscribeTo);
    IF MySelectionContainsSubscriberOrPublisher(isSubcriber) THEN
    BEGIN    {selection contains a single subscriber or publisher}
        IF isSubscriber THEN {selection contains a subscriber}
            SetMenuItemText(menu, iPubSubOptions, gSubOptText)
        ELSE                {selection contains a publisher}
            SetMenuItemText(menu, iPubSubOptions, gPubOptText);
        EnableItem(menu, iPubSubOptions);
    END
    ELSE     {selection contains either no subscribers or publishers }
             { or contains at least one subscriber and one publisher}
        DisableItem(menu, iPubSubOptions);
    IF (gPubCount > 0) OR (gSubCount > 0) THEN
        EnableItem(menu, iShowHideBorders)
    ELSE
        DisableItem(menu, iShowHideBorders);
END;
```

程序清单 3－21 中的过程可以调整编辑菜单中的项目以适应应用程序的文稿窗口。如果应用程序能够取消上次命令动作结果，那么该代码段就可以启用“取消”命令；如果有一个可被剪或复制的选定项，那么该代码段就可以启用“剪”和“复制”命令；如果在剪贴数据存储区中有文字数据，该代码段则可以启用“贴”命令；而且根据当前选定项中是否包含版本或引用程序，该代码段还能够相应地启用与发行和用户程序相关的菜单命令项目。如果当前选定项中包含一个单一引用程序或一个单一发行物，那么应用程序定义的函数 MySelectionContainsSubscriberOrPublisher 将返回 TRUE，否则返回 FALSE。如果 MySelectionContainsSubscriberOrPublisher 函数返回 TRUE，程序清单中的程序将为发行选项（或引用程序选项）命令设定文字并启用该菜单命令项目。如果该函数返回 FALSE，程序清单中的程序将停用发行选项（或引用程序选项）命令。

确定用户是否选择了等效命令组合键

命令的等效组合键允许用户使用键盘来调用一个菜单命令。您可以通过检验事件记录中是否有键按下事件来确定用户是否选择菜单命令的等效组合键。如果用户按下 Command 键的同时又按下另一个字符键，您可以通过使用 MenuKey 函数来判断该组合是否映射到了一个等效命令组合键。

如果您的应用程序支持那些使用除 Command 键之外的其它修正键的等效命令组合键，那么您的应用程序应该检查 modifiers 域并采取相应的动作；您的应用程序是否能够使用 MenuKey 把键映射到菜单命令，完全取决于您使用的修正键。

程序清单 3－22 是一个应用程序定义的过程 DoEvent，它能够确定是否发生了一个键按下事件。如果发生了键按下事件，则可以调用一个由应用程序定义的例程来处理这个键按下事件。

程序清单 3－22　　确定键是什么时候被按下的

```
PROCEDURE DoEvent (event: EventRecord);
BEGIN
    CASE event.what OF
        keyDown, autoKey:      {handle keyboard events}
            DoKeyDown(event);
        {handle other events appropriately}
    END; {of CASE}
END;
```

如果您的应用程序确定用户按键动作，您还需要判断用户是否选择了一个菜单命令的等效组合键。您可以通过检验键按下事件记录的 modifiers 域来做这项工作。如果命令键也被按下，那么您的应用程序应该调用 MenuKey 函数。MenuKey 函数扫描当前菜单列表，寻找一个具有匹配等效命令组合键的菜单命令项目；如果有的话，将返回菜单和菜单命令项目。虽然您不应为多个命令定义同样的等效命令组合键，但 MenuKey 函数可以从右到左扫描菜单，从上到下扫描项目，并返回它找到的第一个匹配的等效命令组合键。

如果您的应用程序除了使用 Command 键外还使用其它等效命令组合键，您可以检验修正键的状态。如果有必要，还可以使用事件管理器函数 KeyTranslate 来帮助将等效命令组合键映射到特定的菜单命令项目。请参阅《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书关于 'KCHR' 资源的讨论，以了解各种键盘组合是如何映射到特定的字符代码的。

程序清单3-23是一个应用程序的DokeyDown过程，它可以处理键按下事件并能够确定用户是否按下了一等效命令组合键。

程序清单 3 - 23　　检验一个键按下事件是不是一个等效命令组合键

```
PROCEDURE DoKeyDown (event: EventRecord);
VAR
     key:           Char;
BEGIN
     key := CHR(BAnd(event.message, charCodeMask));
     IF BAnd(event.modifiers, cmdKey) <> 0 THEN
     BEGIN          {Command key down}
          IF event.what = keyDown THEN
          BEGIN                    {first enable/disable/check }
               MyAdjustMenus;      {menu items properly}
               DoMenuCommand(MenuKey(key));{handle the menu command}
          END;
     END
     ELSE
          MyHandleKeyDown(event);
END;
```

程序清单 3－23 从事件记录的 Message 域中获取被按下的键并通过检验 modifiers 域来判断 Command 键是否也被按下。如果 Command 键也被按下，应用程序将首先调整它的菜单，然后调用一个应用程序定义的过程 DoMenuCommand。DoKeyDown 将 MenuKey 函数返回的值作为一个参数传递给DoMenuCommand过程。

程序清单 3 － 24 为 DoMenuCommand 过程。

响应用户选择的菜单命令项目

您的应用程序可以使用 MenuSelect 函数来确定用户什么时候选择了一个菜单命令，您的应用程序还可以使用 MenuKey 函数来确定用户什么时候按下了一个菜单命令的等效组合键。MenuSelect 和 MenuKey 都返回一个长整数，这个整数表示用户选择的是哪个菜单和菜单命令项目。

MenuSelect 和 Menukey 函数在它们的函数结果的高位字返回菜单的菜单标识符，在低位字返回菜单命令项目编号。如果用户没有选择菜单命令或用户按下的组合键并没有映射您的应用程序菜单的任何等效命令组合键，那么上述函数将在高位字返回 0，低位字的值没有定义。当用户在应用程序或键盘菜单中选择一个项目时，MenuSelect 函数也在高位字返回 0。当用户选定一个您的应用程序增补到帮助菜单的项目时，MenuSelect 函数（和 MenuKey 函数，如果该命令有一个等效命令组合键的话）在高位字返回 kHMHelpMenuID 常数，在低位字返回菜单命令项目。

程序清单 3 - 24 是一个应用程序定义的过程 DoMenuCommand。这个过程依据用户选择的是哪个菜单而采取合适的动作。

在应用程序确定发生了下面两种情况的哪一种，即当光标处于菜单栏时用户按下鼠标按钮（在这种情况下，应用程序调用 MenuSelect 允许用户选择一个命令），或按下 Command 键和另一键（这种情况下应用程序调用 MenuKey 函数），应用程序都会调用 DoMenuCommand 过程。无论是哪种情况，应用程序都将把 MenuSelect 或 MenuKey 返回的函数结果作为一个参数传递给 DoMenuCommand 过程。

程序清单 3 - 24　响应用户的菜单命令选择

```
PROCEDURE DoMenuCommand (menuResult: LongInt);
VAR
    menuID, menuItem: Integer;
BEGIN
    menuID := HiWord(menuResult);   {get menu ID of menu}
    menuItem := LoWord(menuResult); {get menu item number}
    CASE menuID OF
        mApple:
            MyHandleAppleCommand(menuItem);
        mFile:
            MyHandleFileCommand(menuItem);
        mEdit:
            MyHandleEditCommand(menuItem);
        mFont:
            MyHandleFontCommand(menuItem);
        mSize:
            MyHandleSizeCommand(menuItem);
        kHMHelpMenuID:
            MyHandleHelpCommand(menuItem);
        mOutline:
            MyHandleOutlineCommand(menuItem);
        mSubMenu: {user chose item from submenu}
            MyHandleSubLabelStyleCommand(menuItem);
    END;     {end of CASE menuID}
    HiliteMenu(0);{unhighlight what MenuSelect or MenuKey hilited}
END;
```

DoMenuCommand 过程可以调用其它应用程序定义的例程来执行请求的动作。在执行了与所选菜单命令项目相关联的动作之后，您的应用程序应该使用 HiliteMenu 过程取消菜单标题的醒目显示以表明所请求的动作已经完成。

处理苹果菜单

当用户从苹果菜单中选择一个项目时，MenuSelect 函数会在其函数结果的高位字返回应用程序苹果菜单的菜单标识符，在低位字返回所选的菜单命令项目。

如果您的应用程序将一个“关于…”命令作为苹果菜单中的第一个菜单命令项目，而且用户选择了这一项目，那么您就应该显示您的应用程序的“关于…”框。否则，您的应用程序就应该使用 GetMenuItemText 过程来获取菜单命令项目文字，然后调用 OpenDeskAcc 函数，把所选菜单命令项目的文字作为参数传递给该函数。

程序清单 3 - 25 是一个应用程序定义的过程 MyHandleAppleCommand。当用户从苹果菜单中选择了一个项目时，应用程序则调用该过程来响应。

程序清单 3 - 25　　响应用户在苹果菜单对一个项目的选择

```
PROCEDURE MyHandleAppleCommand (menuItem: Integer);
VAR
     itemName: Str255;
     daRefNum: Integer;
BEGIN
     CASE menuItem OF
          iAbout:              {bring up alert for About}
               DisplayMyAboutBox;
          OTHERWISE
          BEGIN {all non-About items in this menu are desktop items, }
                  { for example, DA's, other apps, documents, etc.}
               GetMenuItemText(GetMenuHandle(mApple), menuItem,
                                       itemName);
               daRefNum := OpenDeskAcc(itemName);
          END;
    END; {of CASE}
END;
```

当用户从苹果菜单中选择其它项目而不是“关于”命令项目时，您的应用程序应该调用 OpenDeskAcc。OpenDeskAcc 函数时刻准备打开用户选择的桌面对象；例如，如果用户选择了一个 TeachText 应用程序创建的文稿，OpenDeskAcc 函数安排执行 TeachText 应用程序（或准备打开它，如果还没有打开的话）并返回到您的应用程序。在您的应用程序下次调用 WaitNextEvent 时，它将接收到一个挂起事件，然后进程管理器令 TeachText 成为前台应用程序并指令 TeachText 打开所选文稿。

处理帮助菜单

当用户从帮助菜单中选择一个增补项目时，MenuSelect 和 MenuKey 函数均在高位字返回 kHMHelpMenu 常数（-16490）。增补菜单命令项目的项目编号则通过函数结果的低位字中返回。

程序清单 3-24 中的 DoMenuCommand 过程能够确定用户选择了哪个菜单命令。如果用户从帮助菜单里选择了一个命令，DoMenuCommand 过程将调用应用程序定义的过程 MyHandleHelpCommand。程序清单 3-26 是应用程序定义的过程 MyHandleHelpCommand。这个过程解释了 SurfWriter 应用程序是怎样响应用户选择应用程序帮助菜单中的某一项目的。注意，您应该使用 HMGetHelpMenuHandle 函数，而不是 GetMenuHandle 函数来获取一个应用程序帮助菜单间接指针。

程序清单 3 – 26 用户从帮助菜单中选择命令予以响应

```
PROCEDURE MyHandleHelpCommand (menuItem: Integer);
VAR
     myHelpMenuHdl:                MenuHandle;
     origHelpItems, numItems:      Integer;
     myErr:                        OSErr;
BEGIN
     {get handle to your application's Help menu}
     myErr := HMGetHelpMenuHandle(myHelpMenuHdl);
     IF myErr <> noErr THEN
          EXIT(MyHandleHelpCommand);
     {count the number of items in the Help menu}
     numItems := CountMItems(myHelpMenuHdl);
     origHelpItems := numItems - kNumMyHelpItems;
     IF menuItem > origHelpItems THEN
     BEGIN {user chose an item added by this application}
          {adjust this application's global variables that hold item }
          { numbers of the menu items that this application appended}
          gMyHelpItem1 := origHelpItems +1;
          gMyHelpItem2 := origHelpItems +2;
          MyHelp(menuItem);
     END;
END;
```

苹果电脑公司保留改变帮助菜单中标准项目编号的权力。如果希望确定帮助菜单中的项目编号，可以调用 CountMItems 函数。

处理大小菜单

您的应用程序可以提供一个“大小”菜单以便用户选择各种不同大小的字体。您的“大小”菜单还应为用户提供一种方法，以便用户能够在菜单中随意指定字体大小。例如，您可以提供一个“其它”命令，该将命令显示一个对话框，允许用户选择不同的字体大小。如果用户选择了一个菜单中不存在的字体大小，则可以将一个选择标记添加到“其它”菜单命令，并把所选的“大小”用圆括号括起来添加到“其它”命令的文字中。

您的应用程序应该将那些当前字体能够直接向用户提供的字体大小用轮廓线框起来。对于点阵字体，仅把那些在字体文件夹中实际存在的字体大小用轮廓线框起来即可。对于 TrueType 字体，则应为它支持的所有字体大小加轮廓线。

您的应用程序应该在当前字体大小的菜单命令项目文字旁画一个检查标记，以便向用户指明当前字体大小。如果当前选定项包含一个以上的字体大小，那么就应在选定项包含的每个字体大小名称旁画一个破折号。(3－61页的“更改菜单命令项目的标记符号”解释了怎样向菜单命令项目添加标记和从菜单命令项目中删除标记。)

图 3－35 是一个“大小”菜单，显示了用户使用“其它”命令选择了大小为 31 的新字体大小后的显示。在图 3－35中，大小 9，10，12，18，24 和 36 是由应用程序提供的标准的大小。您的应用程序应在“其它”命令旁画一个选择标记以表明当前字体大小不是标准大小。如果选定项仅包含一个非标准大小，“其它”命令则应在文字“其它”之后说明字体的具体尺寸并用圆括号括起来。在图 3－35 中，当前选定项包含一个非标准的大小 31，所以应用程序就在“其它”命令旁画了一个检查标记并将后面的 31 用圆括号括起来。如果选定项包含多个非标准大小，则在单词“其它”之后用文字“混合”加以说明，并将“混合”用圆括号括起来。如果选定项包含一个或多个标准大小而仅含有一个非标准大小，则在选定项包含的每个标准大小旁画一个破折号，在命令“其它”附近画一个破折号并在命令“其它”文字中将非标准大小用圆括号括起来。

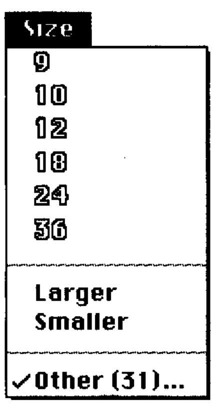


图 3–35　添加了用户指定的字体大小的大小菜单

当用户选择“其它”命令时，您应该在一个对话框中显示当前字体大小并允许用户选择一个新的字体大小。3 - 28 页的图 3 - 16 就是一个示例对话框，一个应用程序在响应用户选择的命令“其它”时可能就会这样显示。

不管当前字体是 TrueType 字体，还是点阵字体，您应该总是用正常字型（像图 3 - 35 中一样）显示命令“其它”的文字，而绝对不要用描边字（空心字）。

程序清单 3 - 27 是一应用程序定义的过程，它处理图 3 - 35 中用户在大小菜单中选择的一个命令项目。

程序清单 3 - 27　　处理大小菜单

```
PROCEDURE MyHandleSizeCommand (menuItem: Integer);
VAR
     numItems:          Integer;
     addItem:           Boolean;
     itemString:        Str255;
     itemStyle:         Style;
     sizeChosen:        LongInt;
BEGIN
     numItems := CountMItems(GetMenuHandle(mSize));
     IF menuItem = numItems THEN
     BEGIN                       {user chose Other command}
          {display a dialog box to allow the user to choose any }
          { size. If the user-specified size is not in the menu, }
          { add a checkmark to the Other command and add the }
          { new font size to the text of the Other command}
        MyDisplayOtherBox(sizeChosen);
     END
     ELSE
     BEGIN
          IF (menuItem = (numItems -2)) OR
             (menuItem = (numItems -3)) THEN
              DoMakeLargerOrSmaller(menuItem, sizeChosen)
          ELSE
          BEGIN               {user chose size displayed in the menu}
               {remove checkmark or dashes from menu items showing }
               { previous size}
               MyRemoveMarksFromSizeMenu;
               {add checkmark to menu item of new current size}
               CheckItem(GetMenuHandle(mSize), menuItem, TRUE);
               sizeChosen := MyItemToSize(menuItem);
               END;
     END;
```

```
{update the document's state or the user's selection as needed}
MyResizeSelection(sizeChosen);
END;
```

如果用户从大小菜单中选择一个项目， MyHandleSizeCommand 过程首先计算菜单中当前的项目数。如果用户选择菜单中的最后一个项目（即“其它”命令），该过程将显示一个类似于 3 - 28 页图 3 - 16 的对话框，让用户选择当前菜单没有列出的“大小”。如果用户选择了一个新的大小，应用程序定义的函数 MyDisplayOtherBox 也将给“其它”命令添加一个选择标记，把新的“大小”添加到“其它”命令的文字中，并在 SizeChosen 变量中返回所选的“大小”。

如果用户在大小菜单中选择了“更大”或“更小”命令，该代码段将调用应用程序定义的例程 DoMakeLargerOrSmaller 来执行请求的动作。如果新的“大小”在菜单中找不到匹配项，DoMakeLargerOrSmaller 过程也会添加一个选择标记并把新的“大小”添加到“其它”命令的文字中。该过程在变量 SizeChosen 中返回所选的“大小”。

如果用户选择了当前菜单中显示的任何“大小”，MyHandleSizeCommand 过程将相应地调整菜单命令项目的标记符号。该代码段将取消先前菜单命令项目的选择标记并将另外一个选择标记添加到用户所选的新“大小”的菜单命令项目中。该代码段使用应用程序定义的函数 MyItemToSize，将所选菜单命令项目的项目编号映射到一个给定的“大小”，并在变量 SizeChose 中返回该“大小”。

然后，该代码段使用应用程序定义的过程 MyResizeSelection 来更新文稿的状态，（如果有用户选定项的话）并将用户的选取的文字尺寸调至所选大小。

从对话框中存取菜单

在 System 7 中，当用户与警告栏或定模式对话框对话时，菜单管理器或您的应用程序允许用户在菜单栏中存取选定的菜单。这样就可以让用户在您的应用程序显示警告栏或定模式对话框的同时作出菜单选择。例如，用户可能希望打开旁述以获得关于怎样响应警告栏的帮助信息。类似地，如果定模式对话框包含几个可编辑的文字域，用户可能会发现从一个文字域复制文字并把它贴到另一个域中是比较简单的。图3-36 给出了一个包含可编辑文字域的定模式对话框。注意，此时只有编辑和帮助菜单是有效的，其它所有菜单均已被停用。这就使得用户能够存取编辑命令和旁述。

附　注

在 System 6 中，未经您的应用程序特别许可，严禁用户从警告栏或定模式对话框中存取菜单栏中的菜单。例如，在 System 6 中，如果您要在定模式对话框中支持标准编辑菜单命令的等效命令组合键，您的应用程序必须提供一个过滤程序以取代标准的过滤过程。在 System 7 中，您可以使用菜单管理器启用这些命令。



图 3-36 从定模式对话框中存取菜单

在显示不定模式或可移动定模式对话框时，您的应用程序应依据对话框对菜单作适当的调整。例如，当可移动定模式对话框处在窗口最前面时，您的应用程序应该启用苹果菜单，如果对话框包含可编辑的文字项目，那么还应该启用编辑菜单，如果有必要还可以启用或停用任何其它菜单，如果在显示对话框的同时用户不能执行应用程序添加到帮助菜单的那些项目，您的应用程序应停用这些项目。

当应用程序显示警告栏时，系统软件将自动地停用除帮助菜单以外的所有菜单（在帮助菜单中除显示旁述／隐藏旁述命令外，所有其它项目也都置无效）。

在显示定模式对话框时，您的应用程序也应该相应地启用和停用其菜单。例如，如果对话框包含可编辑的文字项目，就应该启用编辑菜单，如果在显示对话框的同时用户不能执行应用程序添加到帮助菜单的那些项目，则应停用这些项目。如果您的应用程序在定模式对话框中处理对菜单栏的存取，它应停用苹果菜单或苹果菜单中的第一个项目。

如果您的应用程序并不专门在从警告栏或定模式对话框中处理对菜单栏的存取，那么在有些情况下菜单管理器会自动地为您停用相应的菜单，下面的段落中有相关的说明。

当您的应用程序显示警告栏或定模式对话框（即dBoxProc类型的窗口）时，菜单管理器（与对话管理器一起）总是适当地调整经过系统处理的菜单并执行以下动作:

1. 除显示旁述（或隐藏旁述）命令外，停用帮助菜单的所有菜单命令项目。

2. 停用应用程序菜单的所有菜单命令项目。

3. 如果菜单栏中有键盘输入菜单，除了禁止的“About Keyboard（关于键盘）…”命令外，可以启用其它的键盘菜单命令。

另外，如果您的应用程序接着调用 ModalDialog 过程，菜单管理器（与对话管理器一起）将执行两个其它的动作:

4. 停用应用程序的所有菜单。

5. 如果定模式对话框包含可编辑文字域，则会启用标准等效命令组合键为Command-X，Command-C和Command-V。然后，用户可以使用菜单命令或者它们的等效组合键来剪、复制和贴文字。(等效命令组合键为 Command-X的菜单命令项目必须是前5个菜单命令项目之一。)

当用户取消定模式对话框时，菜单管理器将还原所有菜单，使它们恢复到定模式对话框出现前的状态。

有时当您调用ModalDialog时，动作4和5并不发生。如果下列条件的任何一个成立，在步骤4和5中涉及的启用和停用将不会发生。

- 您的应用程序没有苹果菜单
- 您的应用程序有苹果菜单，但当定模式对话框显示时该菜单被禁止使用
- 您的应用程序有菜果菜单，但当定模式对话框显示时该菜单的第一个项目被停用

附　注:

如果您的应用程序已经处理了在定模式对话框中对菜单栏的存取并且您不希望通过System 7自动启用或停用菜单，那么在显示一个定模式对话框时，您应该保证这些条件中的一个或多个成立。

当您的应用程序显示没有可编辑文字项目的警告栏或定模式对话框时，您的应用程序可以允许系统软件处理菜单栏存取。在其它情况下，您的应用程序应该处理它自己的菜单栏存取。

当您显示可移动定模式对话框和不定模式对话框时，系统软件总是保留帮助、键盘输入和应用程序菜单以及它们的命令项目以便您使用。对于这些对话框类型，您必须停用相应的菜单并依据它们的内容适当地处理菜单栏存取。

当您的应用程序显示可移动的定模式对话框（ movableDBoxProc 类型的窗口）时，您的应用程序没有必要调整系统处理的菜单，但应该停用除苹果菜单外的所有其它菜单以及编辑菜单(如果可移动定模式对话框包含可编辑文字项目)。让苹果保持激活状态可以为您的应用程序用它打开其它应用程序提供方便，而让编辑菜单保持激活状态则可以为用户在可编辑文字项目中使用剪、复制和贴命令提供方便。(您也可以启用“取消”和“清除”命令；如果不然，而停用编辑菜单中的所有其它命令。)

当您的应用程序取消可移动定模式对话框、不定模式对话框或有可编字文字项目的定模式对话框时，它必须把它在显示对话框前停用的任何菜单还原至它们原先的状态。请参阅本书“对话管理器”关于对话框的补充说明。

编写您自己的菜单定义过程

菜单管理器可以使用菜单定义过程和菜单栏定义函数，在菜单和菜单栏上显示和执行基本的操作。菜单定义过程可以执行菜单里所有菜单命令项目的绘制工作，还可以执行各类菜单间互不相同的所有动作。菜单栏定义函数负责绘制菜单栏，当用户在菜单间移动光标时，它还执行与菜单显示相关的大部分绘制活动。

苹果提供了一个标准的菜单栏定义函数，作为一个资源储存在系统文件里。标准的菜单栏定义过程是'MBDF'资源，资源标识符为 0。在创建菜单和菜单栏时，菜单管理器可以通过系统设定的标准菜单栏定义函数来管理它们。尽管菜单管理器允许您提供自己的菜单栏定义函数，但苹果还是建议您使用标准的菜单栏定义函数。

菜单管理器使用标准的菜单栏定义函数可以完成下列任务:

- 绘制菜单栏
- 清除菜单栏
- 确定光标是否处在菜单栏或任何当前显示的菜单中
- 计算菜单标题的左边缘
- 醒目显示菜单标题
- 倒置整个菜单栏
- 抹除菜单的背景色并绘制菜单的结构（阴影）
- 存储或恢复菜单后面的位

苹果提供了一个标准的菜单定义过程，作为一种资源储存在系统文件中。标准的菜单定义过程是'MDFE'资源，其资源标识符为 0。标准的菜单定义过程处理3类菜单：下拉式、弹出式和嵌套式；它还可以在菜单中实现滚动。在定义您的菜单时，您可以指定菜单定义过程，菜单管理器在管理菜单时要使用该过程。在您的应用程序中，一般应使用标准定义过程。但是，如果您需要某种标准菜单定义过程没有提供的功能（例如，如果您要在您的菜单中包括更多的图形），您可以编写自己的菜单定义过程。

菜单管理器可以使用标准的菜单定义过程来完成以下功能:

- 计算菜单的尺寸
- 在菜单中绘制菜单命令项目
- 当用户在菜单间移动光标时，醒目显示菜单命令项目和取消菜单命令项目的醒目显示
- 确定用户从菜单中选择了哪个命令项目

如果您提供了自己的菜单定义过程，它也应执行这些任务。另外，您的菜单定义过程还应支持菜单滚动、菜单色彩以及旁述。

如果您提供了自己的菜单定义过程，就应把它储存在一个'MDEF'类型的资源中，在每个使用您自己的菜单定义过程的菜单说明中包括其资源标识符。如果使用 GetMenu（或 GetNewMBar）来创建一个菜单，菜单管理器将把菜单定义过程读入存储器，并在该菜单记录的 MenuProc 域中储存该过程的间接指针。

当您的应用程序使用 GetMenu（或 GetNewBar）来创建一个使用您的菜单定义过程的新的菜单时，菜单管理器将为该菜单创建一个菜单记录，并根据菜单的资源描述填入了 menuID，menuProc，enableFlags，和 menuData 域。菜单管理器还将读入每个菜单命令项目的数据并把它作为变量数据储存在菜单记录的末尾。菜单定义过程负责解释数据的内容。例如，在从 3-151 页的“菜单资源”中有关于标准菜单定义过程解释这种数据的说明。在读入 'MDEF' 菜单资源后，菜单管理器请求菜单定义过程计算菜单的大小，并把这些值保留在该菜单记录的menuWidth和menuHight域中。

注意，绘制菜单时，菜单管理器首先请求您的菜单定义过程计算该菜单的尺寸（菜单矩形）。然后，菜单管理器请求菜单栏定义函数绘制菜单的结构（阴影），并抹除菜单内容使之变成背景色。最后，菜单管理器请求您的菜单定义过程在菜单中绘制项目。当用户将光标移入或移出菜单命令项目时，菜单管理器将请求您的菜单定义过程相应地醒目显示项目或取消项目的醒目显示。您的菜单定义过程还应确定何时将一滚动指示符添加到菜单，当光标处在一个滚动项目中时能相应地滚动菜单。您的菜单定义还负责显示和取消与菜单命令项目相关的任何旁述。

当菜单管理器请求您的菜单定义过程执行菜单上的动作时，它将为您的过程提供其菜单记录间接指针。这样，您的过程就可以存取菜单记录中的数据，并使用菜单记录的变量数据分区中的数据来正确地处理菜单命令项目。但您的菜单定义过程不应假定 A5 寄存器已被正确设置，所以您的过程不能参照任何 QuickDraw 全局变量。

菜单管理器将在Message参数中向您的菜单定义过程传递一个值，指出您的菜单定义过程应该执行的动作。菜单管理器总是传递一个菜单的菜单记录间接指针，在 theMenu 参数中的操作会影响该菜单。菜单管理器还将在其它参数中传递辅加信息，具体传递什么信息则由所请求的动作决定。

程序清单 3-28 给出了您可能说明菜单定义过程的方法。

程序清单 3-28　　一个示例菜单定义过程

```
PROCEDURE MyMDEF ( message: Integer; theMenu: MenuHandle;
                    VAR menuRect: Rect; hitPt: Point;
                    VAR whichItem: Integer);

{any support routines used by the main program of your MDEF }
{ go here}

BEGIN
     CASE message OF
          mDrawMsg:
               MyDrawMenu(theMenu, menuRect);
          mChooseMsg:
               MyChooseItem(theMenu, menuRect, hitPt, whichItem);
          mSizeMsg:
               MySizeTheMenu(theMenu);
          mPopUpMsg:
               MyCalcMenuRectForOpenPopUpBox(theMenu, hitPt, menuRect);
     END;
END;
```

下一节将详细介绍当您的菜单定义过程在 message 参数中收到 mDrawMsg、mChoose-Msg 或 mSizeMsg 常数时，它应该如何响应。有关菜单定义过程和菜单管理器给您的过程传送参数的全面信息，请参阅 3-148 页的“菜单定义过程”。

计算菜单的尺寸

无论什么时候菜单管理器创建了一个菜单或需要计算您的菜单定义过程管理的一个菜单的大小，菜单管理器都可以调用您的过程，并在 message 参数中指定 mSizeMsg 常数，请求您的过程来计算该菜单的大小。

3-90 页的程序清单 3-29 是一个应用程序定义的支持例程 MySizeTheMenu，应用程序的菜单定义过程要使用这个支持例程。在计算了菜单矩形的高度和宽度后，菜单定义过程便将得出的数值分别储存在该菜单的菜单记录的 menuWidth 和 menuHeight 域中。

程序清单 3-29 计算一个菜单的大小

```
PROCEDURE MySizeTheMenu(theMenu: MenuHandle);
VAR
    itemDataPtr:        Ptr;
    numItems:           Integer;
BEGIN
    HLock(Handle(theMenu));
    WITH theMenu^^ DO
    BEGIN    {menuData points to title of menu and additional item data}
        itemDataPtr := @menuData;
        {skip past the menu title}
        itemDataPtr := POINTER(ORD4(itemDataPtr)+ itemDataPtr^ +1);
    END;
    numItems := CountMItems(theMenu);
    {calculate the height of the menu--each item's height can vary }
    { according to whether the item has an icon or a script code defined. }
    { The height of the menu should not exceed the height of the }
    { screen minus the menu bar height. }
    { Store the height in the menu's menu record}
    theMenu^^.menuHeight := MyCalcMenuHeight(itemDataPtr, numItems);

    {calculate the width of the menu (the width of the longest item): }
    { for each item calculate the width as }
    { width = iconWidth + markWidth + textWidth + subMenuWidth }
    {         + cmdKeyComboWidth }
    { If an item doesn't have a characteristic, use 0 as the width of }
    { that characteristic. }
    { To calculate the width of item's text, must consider script code and }
    { width of the font. }
    { The width of the menu should not exceed the right or left }
    { boundaries of the screen. }
    { Store the width in the menu's menu record}
    theMenu^^.menuWidth := MyCalcMenuWidth(itemDataPtr, numItems);
    HUnLock(Handle(theMenu));
END;
```

绘制菜单中的菜单命令项目

当光标在菜单定义过程管理的菜单的菜单标题上时，只要用户按下鼠标按钮，菜单管理器就会调用菜单栏定义函数来醒目显示菜单标题，绘制菜单的结构，并抹除菜单的内容，令其变成背景色。然后，菜单管理器调用您的菜单定义过程并在 message 参数中指定 mDrawMsg 常数，请求您的过程绘制菜单命令项目。当您的菜单定义过程

收到这个常数时，它应该在menuRect参数指定的矩形中绘制由theMenu参数指定的菜单的菜单命令项目。菜单管理器在调用您的菜单定义过程前，应将当前图形端口设定为窗口管理器端口。您的菜单定义过程能够通过检验菜单记录中的数据，确定如何绘制菜单命令项目。

如果您的菜单定义过程支持彩色菜单，那么您的过程应该核查应用程序的菜单色彩信息表，以显示用来绘制每个项目的色彩。如果应用程序菜单色彩信息表中包含了一个项目的彩色输入，那么就可以使用该彩色绘制此项目。如果表中没有特定项目的彩色输入项，就应使用菜单标题输入中定义的系统设定项目色彩。如果菜单标题输入不存在，则应使用菜单栏输入中定义的系统设定项目色彩。如果菜单栏输入不存在，则使用白的底黑色绘制该项目。

如果您的菜单定义过程支持滚动菜单，那么（如果有必要）它就应该在绘制菜单命令项目时插入滚动指示符。

程序清单3-30给出了一个应用程序定义的支持例程 MyDrawMenu，应用程序的菜单定义过程将使用该例程。MyDrawMenu过程将根据项目的定义特征来绘制菜单中的每个项目。停用项目应该使用GetGray函数返回的色彩来绘制。将bkgnd参数中项目背景的 RGB色彩传送给GetGray函数；传送fgnd参数中项目启用文字的RGB色彩。如果这两种指定的色彩间存在可用的色彩，那么GetGray函数返回TRUE，并将您应该用来绘制该项目的色彩返回到fgnd参数中。

程序清单 3-30　　绘制菜单命令项目

```
PROCEDURE MyDrawMenu(theMenu: MenuHandle; menuRect: Rect);
VAR
    numItems:       Integer;
    itemRect:       Rect;
    item:           Integer;
    currentOffset: LongInt;
    nextOffset:     LongInt;
BEGIN
    numItems := CountMItems(theMenu);
    currentOffset := 0;
    nextOffset := 0;
    FOR item := 1 TO numItems DO
    BEGIN
    {calculate the enclosing rectangle for this item}
        itemRect := MyCalcItemRect(item, menuRect, currentOffset, nextOffset);
    {draw the item--index into the item-specific data from the menu record }
    { to get the characteristics of this menu item and draw the item }
```

```
    { according to its defined characteristics. For example, draw the item's }
    { text in its defined style & font of its defined script, draw any icon, }
    { mark, submenu indication, or keyboard equivalent, and draw each }
    { characteristic of the item according to its color entry in the menu's }
    { menu color information table. }
    { Draw disabled items in gray--use the GetGray function to return the }
    { appropriate color. Also draw dividers using the gray color }
    { returned by GetGray}
        MyDrawTheItem(item, itemRect, menuRect, currentOffset);
    END;
    {if your menu supports scrolling, insert scrolling indicators if needed}
    MyInsertScrollingArrows(menuRect);
END;
```

确定光标是否位于启用菜单命令项目

当用户将光标拖拽入或拖拽出菜单定义过程管理的显示菜单的菜单命令项目时，菜单管理器调用您的过程，并在message参数中指定mChooseMsg常数，请求您的过程确定光标是否在一个菜单命令项目中，另外还请求您的过程适当地醒目显示菜单命令项目或取消菜单命令项目的醒目显示。当您的菜单定义过程获得这个常数后，它应该使用在menuRect参数中指定的矩形、在hitPt参数中指定的鼠标位置以及在whichItem参数中指定的项目编号，来确定所采取的正确动作。

如果要显示用户是否选择了一个启用的项目，您的菜单定义过程应该确定鼠标位置是否在menuRect参数指定的矩形内。如果是，它应该核查菜单的有效状态。如果菜单是有效用的，那么您的菜单定义过程还应该确定hitPt参数中指定的鼠标位置所有的项目是不是一个有效的菜单命令项目。

如果鼠标位于一个有效的菜单命令项目中，那么您的菜单定义过程应该取消whichItem参数指定项目的醒目显示，醒目显示新的项目，并向whichItem返回新的项目编号。

如果鼠标位置不是在一个启用的菜单命令项目中，那么您的菜单定义过程应该取消whichItem参数指定项目的醒目显示，并向whichItem参数返回 0。

当您的菜单定义过程在彩色菜单中绘制一个醒目显示状态的菜单命令项目时，它应该颠倒背景色和项目色彩（即反白显示），然后绘制菜单命令项目。当您的菜单定义过程需要将一个菜单项目变回到它的正常（非醒目显示）的状态时，它应该复原该菜单命令项目的背景色和项目色彩并绘制该菜单命令项目。

如果您的菜单定义过程支持滚动菜单，当用户将光标移入指示符区时，或者当光标直接在菜单的上面或下面时，它应该滚动该菜单。若用户能够往下滚动菜单（通过将光标拖拽过最后一个项目，以便显示更多的项目），应在菜单中最后项目的地方放置

一个向下的三角形指示符。若用户能够往上滚动菜单(通过将光标拖拽过第一个项目,以便显示原来在菜单顶部的项目),应在菜单中第一个项目的地方放置一个向上的三角形指示符。

每当醒目显示一个新项目时,您的菜单定义过程都应该适当地为所有的菜单设定全局变量 MenuDisable。不论上次选择的菜单命令项目是否停用,都应将 MenuDisable 设定为该项目的菜单标识符和项目编号。MenuChoice 函数使用 ModalDialog 中的值来确定一个选定菜单命令项目是否停用。

程序清单 3-31 给出了一个应用程序定义的支持例程 MyChooseItem,应用程序的菜单定义过程将使用该例程。这个例程能够确定(如果有的话),hitPt 参数指定的点在哪一个项目中。如果该点是在和上一个项目不同的一个启用项目中,MyChooseItem 过程将取消旧项目的醒目显示,并醒目显示新项目。但是,如果该项目点在分割线或一个停用的项目中,那么 MyChooseItem 过程不醒目显示新的项目。

该过程还将适时取消旁述,并在帮助汽泡被打开时,显示任何新项目(除了分割线或滚动指示符以外的所有项目)的帮助汽泡。MyChooseItem 过程将新项目的项目编号返回 whichItem 参数,如果没有选择任何项目,则返回 0。即使在列表中没有显示,如果项目是一个停用项目,过程也会在 whichItem 参数中返回 0,并将 MenuDisable 全局变量设定为停用项目的菜单标识符和项目编号。

程序清单 3-31　选择菜单命令项目

```
PROCEDURE MyChooseItem  (theMenu: MenuHandle; menuRect: Rect; hitPt: Point;
                         VAR whichItem: Integer);
VAR
    oldWhichItem:       Integer;
    MenuChoicePtr:          ^LongInt;
    numItems, item, max:  Integer;
    itemChosen:           Integer;
    inScroll:             Integer;
    currentOffset:        LongInt;
    nextOffset:           LongInt;
BEGIN
    oldWhichItem := whichItem;
    whichItem := 0;
    itemChosen := 0;
    MenuChoicePtr := POINTER(kLowMemMenuDisable);
    numItems := CountMItems(theMenu);
    {find out whether the hitPt is in an item's rectangle, and if so, }
    { determine which item}
```

```
item := 1;
max := numItems + 1;
currentOffset := 0;
nextOffset := 0;
REPEAT
    itemRect := MyCalcItemRect(item, menuRect, currentOffset, nextOffset);
    IF PtInRect(hitPt, itemRect) THEN {hitPt is in this item}
       itemChosen := item;
    item := item + 1;
UNTIL (item = MAX) OR (itemChosen <> 0);
IF itemChosen = 0 THEN
BEGIN {the mouse isn't in any item of this menu;unhighlight previous item}
    MyNotInMenu(menuRect, oldWhichItem);

END
ELSE
BEGIN     {the mouse is in this menu item. }
          { First see if a previous item was highlighted}
    IF ((oldWhichItem <> 0) AND (oldWhichItem <> itemChosen)) THEN
    BEGIN
          {a previous item was highlighted--unhighlight it}
          itemRect := MyCalcOldItemRect(oldWhichItem, menuRect);
          IF HMGetBalloons THEN    {if Balloon Help is on then }
               HMRemoveBalloon;{ remove any balloon that might be showing}
          MyHighlightItem(itemRect, oldWhichItem, FALSE);
    END;
    IF HMGetBalloons and MyIsItemDivider(itemChosen) THEN
          {Balloon Help is on and item is divider}
          HMRemoveBalloon;    {remove any balloon that might be showing}
    IF MyIsItemEnabled(itemChosen) THEN
    BEGIN
          {the item is enabled, so highlight the item the cursor is in}
          itemRect := MyCalcNewItemRect(itemChosen, menuRect, currentOffset);
          {the highlighting routine must also support scrolling correctly }
          { (if the cursor is in a scrolling item, don't highlight the item)}
          inScroll := MyIsScrollItem(itemChosen);
          MyHighlightItem(itemRect, itemChosen, inScroll);
          IF HMGetBalloons AND inScroll THEN
               HMRemoveBalloon {remove any balloon that might be showing}
          ELSE
          BEGIN {display help balloon for this item, if any}
               IF HMGetBalloons THEN
```

```
    BEGIN
         IF StillDown THEN {mouse button is still down in this item}
           {this routine sets up the needed parameters and then }
           { calls HMShowMenuBalloon}
            MyShowMenuBalloon(itemChosen, itemRect);
    END;
   END;
  END;
 END;
END;
```

菜单管理器参考资料

本节介绍了菜单管理器的数据结构和例程。另外还介绍了各种资源,包括您可以用来创建菜单和菜单栏的资源 'MBAR' 和 'MENU' 。

数据结构

本节介绍了菜单记录、菜单列表和菜单色彩信息表。菜单管理器在菜单记录中保留您的应用程序中菜单的信息。菜单管理器在一个称为菜单列表的数据结构中保留菜单栏中所有菜单的信息。

菜单管理器在一个菜单色彩信息表中储存您的应用程序菜单的色彩信息。如果您的菜单栏或菜单需要使用和系统设定色彩不同的色彩,那么您可以为应用程序的菜单色彩信息表添加项目。您可以使用 SetMCentries 过程或通过提供‘mctb’资源为该表添加项目。

菜单记录

菜单记录包含单个菜单的信息。您的应用程序不应该操纵和存取一个菜单记录的域而应该使用菜单管理器的例程来创建和管理应用程序中的菜单。若要引用一个菜单,应使用菜单的菜单记录间接指针。

MenuInfo数据类型定义了菜单记录。MenuHandle数据类型是一个菜单记录的句柄。

```
TYPE  MenuPtr      = ^MenuInfo;  {pointer to a menu record}
      MenuHandle   = ^MenuPtr;   {handle to a menu record}
```

下面是菜单记录的结构：

```
TYPE MenuInfo =                        {menu record}
    RECORD
        menuID:         Integer;   {number that identifies the menu}
        menuWidth:      Integer;   {width (in pixels) of the menu}
        menuHeight:     Integer;   {height (in pixels) of the menu}
        menuProc:       Handle;    {menu definition procedure}
        enableFlags:    LongInt;   {indicates whether menu and }
                                   { menu items are enabled}
        menuData:       Str255;    {title of menu}
        {itemDefinitions}          {variable-length data that }
                                   { defines the menu items}
    END;
```

菜单记录的域说明

menuID　是用来标识菜单的一个数字。应用程序的每个菜单都必须有一个唯一的菜单标识符。当您创建菜单时，您的应用程序会为指定一个菜单标识符。您可以借助菜单标识符和 GetMenuHandle 函数为菜单的菜单记录获取一个间接指针。在定义嵌套式菜单时，您必须使用 1 到 235 之间的数为应用程序子菜单指定菜单标识符数；使用 236 到 255 之间的数为桌面附件指定子菜单。

menuWidth　菜单的水平尺寸，以像素为单位。

menuHeight　菜单的垂直尺寸，以像素为单位。

menuProc　菜单的菜单定义过程间接指针。菜单管理器使用该菜单定义过程来绘制菜单。

enableFlags　表示菜单标题和菜单中前 31 个项目的启用状态的值。第 31 个项目以后的所有菜单项目都是通过系统设定来启用的，而且只能通过停用整个菜单来停用这些项目。

menuData　定义菜单标题的一个字符串。虽然 menuData 域是由 MenuInfo 数据结构中的Str255数据类型来定义的，但是菜单管理器只分配菜单标题所必需的存储空间：标题字符串中的字符数加 1。

itemDefinitions　长度可变的数据，用来定义菜单中每个菜单命令项目的特征。如果菜单使用标准的菜单定义过程，那么该数据在概念上可以作如下定义：

```
itemData: ARRAY[1..X] OF
    itemString:   String;   {text of menu item}
    itemIcon:     Byte;     {icon number minus 256}
```

```
itemCmd:      Char; {keyboard equivalent or }
                    { value ($1B) indicating }
                    { item has a submenu, or }
                    { ($1C) if item has }
                    { a script code, or }
                    { ($1D) if item's 'ICON' }
                    { should be reduced, or }
                    { ($1E) if item has an }
                    { 'SICN' icon}
itemMark:     Char; {marking character or }
                    { menu ID of submenu}
itemStyle:   Style; {style of menu text}
endMarker:    Byte; {contains 0 if no }
                    { more menu items}
```

菜单定义过程保留菜单命令项目的信息。通常，您应在'MENU'资源中定义菜单命令项目，菜单管理器在菜单的菜单记录中储存项目的说明信息。

您的应用程序不应该直接更改菜单记录中任一域的值。而应该使用菜单管理器例程来更改菜单命令项目的特征或者对菜单作其它修改。

菜单列表

菜单列表包含菜单栏中菜单的信息、子菜单的信息以及弹出式菜单的信息。菜单列表包含0个、1个或更多个菜单的菜单记录的间接指针，并包含菜单管理器用来管理菜单的其它信息。

使用InitMenu过程可以创建一个应用程序的当前菜单列表。当前菜单列表包含当前在菜单栏中的所有菜单的菜单记录的间接指针，以及应用程序插入该菜单列表的所有子菜单或弹出式菜单的菜单记录的间接指针。菜单栏按顺序显示菜单列表中的所有菜单(子菜单和弹出式菜单除外)。

InitMenu创建的初始菜单列表不包含任何菜单的间接指针。在将菜单添加到菜单列表，或从菜单列表中删除时，菜单管理器动态地在菜单列表中分配存储空间。

您的应用程序不应该直接在菜单列表中改变或存取信息。您应该使用菜单管理器例程来创建一个菜单列表，并将菜单添加到当前菜单列表中或从当前菜单列表中删除。

通常，您应该在'MBAR'资源中定义应用程序的菜单栏，并使用GetNewMBar函数创建一个菜单列表。GetNewMBar函数为菜单列表返回一个间接指针。您可以使用SetMenuBar过程将当前菜单列表设定为GetNewMBar返回的菜单列表。

菜单列表的结构是菜单管理器专用的。但是，为了便于理解概念，下面定义了菜单列表的一般结构。

```
TYPE DynamicMenuList =
    RECORD
        lastMenu:      Integer;      {offset to last pull-down menu}
        lastRight:     Integer;      {pixel location of right edge }
                                     { of rightmost menu in menu bar}
        mbResID:       Integer;      {upper 13 bits are the resource ID of menu }
                                     { bar defn function, low 3 bits the variant}
        menu:          ARRAY[1..X]   {variable array with one record for }
                       OF MenuRec;   { each menu}
        lastHMenu:     Integer;      {offset to last submenu or pop-up menu}
        menuTitleSave:               {handle to bits behind inverted menu title}
                       pixMapHandle;
        hMenu:         ARRAY[1..Y]   {variable array with one record for }
                       OF HMenuRec;  { each submenu or pop-up menu}
      END;
```

菜单管理器能够动态分配记录，这些记录中包含有菜单栏中的菜单、子菜单和弹出式菜单的菜单记录的间接指针。这些记录在概念上可以定义为 MenuRect 和 HMenuRec 数据类型。菜单管理器可以使用类似于 MenuRect 数据类型的数据结构，在菜单列表中储存有关下拉式菜单的信息。

```
TYPE MenuRec =
    RECORD
        menuOH:   MenuHandle;    {handle to menu's menu record}
        menuLeft:Integer;        {pixel location of left edge }
                                 { of this menu}
    END;
```

菜单管理器以类似于HMenuRec数据类型的数据结构，在一个菜单列表的结尾储存子菜单和弹出式菜单的信息。

```
TYPE HMenuRec =
    RECORD
        menuHOH: MenuHandle;   {handle to menu's menu record}
        reserved:     Integer;      {reserved}
    END;
```

菜单色彩信息表记录

您的应用程序的菜单色彩信息表定义了菜单栏、菜单标题、菜单命令项目的文字和特征，以及显示菜单的背景色的标准色彩。如果您不向该表添加任何输入，那么菜

单管理器将使用系统设定色彩即黑色和白色绘制您的菜单。您可以通过向应用程序的菜单色彩信息表添加输入来为您的菜单添加色彩。为此，您可以使用菜单管理器例程定义这些项目或者在 'mctb' 资源中定义这些输入。注意，菜单色彩信息表使用了不同于标准色彩表的格式。

菜单管理器将有关应用程序的菜单色彩信息表的信息作为一个菜单彩色输入记录的阵列来保留。

```
TYPE  MCTable = ARRAY[0..0] OF MCEntry;        {menu color table}
      MCTablePtr = ^MCTable;       {pointer to a menu color table}
      MCTableHandle = ^MCTablePtr;{handle to a menu color table}
```

菜单色彩输入是用 MCEntry 数据类型来定义的。

```
TYPE MCEntry =                          {menu color entry}
    RECORD
        mctID:          Integer; {menu ID or 0 for menu bar}
        mctItem:        Integer; {menu item number or 0 for }
                                 { menu title}
        mctRGB1:        RGBColor;{usage depends on mctID and }
                                 { mctItem}
        mctRGB2:        RGBColor;{usage depends on mctID and }
                                 { mctItem}
        mctRGB3:        RGBColor;{usage depends on mctID and }
                                 { mctItem}
        mctRGB4:        RGBColor;{usage depends on mctID and }
                                 { mctItem}
        mctreserved:  Integer; {reserved}
    END;

    MCEntryPtr = ^MCEntry;      {pointer to a menu color entry}
```

菜单彩色项目记录的前两个域 mctID和mctItem，可以将输入定义为菜单栏输入项、菜单标题输入或菜单命令项目输入。接下来的 4 个域指定了 mctID 和 mctItem 域说明的任何类型输入的色彩信息。菜单色彩信息表最后一个输入的 mctID 域的值是 -99，最后一个输入的其它域是保留的。菜单管理器自动在菜单色彩信息表中创建最后输入；如果您希望为一个菜单添加菜单色彩输入，那么您的应用程就序不应该使用值 -99 作为它的菜单标识符。

当您的应用程序首次调用 InitMenus 或 InitProcMenu 时，菜单管理器创建它的菜单色彩信息表。它创建的菜单色彩信息表在开始时（除了表示该表结尾的最后输入外）是空的。

表 3-7 介绍了菜单管理器是如何解释菜单色彩信息表中每个类型的菜单彩色输入项的 mctID 和 mctItem 域的。

表 3-7 菜单栏输入的彩色信息

	标识符	项目	RGB1	RGB2	RGB3	RGB4
菜单栏	0	0	系统设定菜单标题色彩	菜单的系统设定背景色	系统设定的项目	系统设定栏彩色色彩
菜单标题	N<>0	0	菜单标题栏色彩	色彩	系统设定项目	菜单的背景色彩
菜单项目	N<>0	M<>0	标记彩色	项目文字色彩	等效命令组合键色彩	菜单的背景色
最后输入	−99	保留	保留	保留	保留	保留

菜单栏输入由菜单色彩输入记录定义，它的 mctID 和 mctItem 域均包含 0。在一个菜单色彩信息表中，您只能定义一个菜单栏输入。若您不为应用程序的菜单色彩信息表提供一个菜单栏输入，菜单管理器就使用标准的菜单栏彩色（白底黑字），它对其它菜单元素也使用标准色彩。您可以提供一个菜单栏输入，为菜单标题、显示菜单的背景、菜单中的项目和菜单栏指定系统设定色彩。菜单栏输入的色彩信息域解释如下:

- mctRGB1 指定菜单标题的系统设定色彩。如果一个菜单没有菜单标题输入，菜单管理器就会使用该域的值作为菜单标题的色彩。
- mctRGB2 指定显示菜单背景的系统设定色彩。如果一个菜单没有菜单标题输入，菜单管理器就会使用该域的值作为菜单显示时的背景色。
- mctRGB3 指定显示菜单中项目的系统设定色彩。如果一个菜单命令项目没有菜单命令项目输入，或没有在菜单标题输入中定义系统设定色彩，菜单管理器就会使用该域的值作为菜单命令项目的色彩。
- mctRGB4 指定菜单栏的系统设定色彩。如果一个菜单没有菜单栏输入（以及没有任何菜单标题输入），菜单管理器便在菜单栏中使用标准彩色。

菜单标题输入由菜单色彩输入记录定义，它的 mctID 域包含一个菜单标识符，mctItem 域包含 0。 您对每个菜单只能定义一个菜单标题输入。如果您不在应用程序的菜单色彩信息表为一个菜单提供菜单标题输入，那么菜单管理器就会使用菜单栏输入定义的彩色。如果菜单栏输入不存在，菜单管理器就使用标准彩色（白底黑字）。您可以提供一个菜单标题输入，为给定菜单的标题和背景指定一个色彩，并为它的项目指定一种系统设定色彩。菜单标题输入的色彩信息域解释如下:

- mctRGB1 指定给定菜单的菜单标题的彩色。如果一个菜单没有菜单标题输入，菜单管理器就会使用菜单栏输入中定义的系统设定值
- mctRGB2 指定菜单栏的系统设定色彩。如果一个菜单色彩信息表没有菜单栏输入，菜单管理器就会使用该域的值作为菜单栏的彩色。如果菜单栏输入已经存在，菜单管理器则用菜单栏输入的 mctRGB4 域中定义的值代替菜单标题输入的 mctRGB2 域中的值
- mctRGB3 指定显示菜单命令项目的系统设定色彩。如果一个菜单命令项目没有菜单命令项目输入，或没有在菜单标题输入中定义系统设定色彩，菜单管理器便使用该域的值作为菜单命令项目的色彩
- mctRGB4 指定菜单栏菜单背景色

菜单命令项目输入由菜单栏色彩输入记录定义，它在mctID域包含一个菜单标识符，在 mctItem 域包含一个项目编号。您只能为每个菜单命令项目定义一个菜单命令项目输入。如果您不在应用程序的菜单色彩信息表中为项目提供菜单命令项目输入，那么菜单管理器就会使用菜单标题输入定义的色彩（如果菜单包含的项目没有一个菜单标题输入，就使用菜单栏输入提供的色彩）。如果菜单标题输入和菜单栏输入都不存在，菜单管理器便用黑色绘制标记、文字和等效命令组合键。您可以通过提供一个菜单命令项目输入来指定所给菜单命令项目的标记、文字和等效命令组合键的彩色。菜单命令项目输入的色彩信息域解释如下：

- mctRGB1 指定菜单命令项目标记的彩色。如果一个菜单命令项目没有菜单命令项目输入，菜单管理器就使用菜单标题输入或菜单栏输入中定义的系统设定值
- mctRGB2 指定菜单命令项目文字的彩色。如果一个菜单命令项目没有菜单命令项目输入，菜单管理器就使用菜单标题输入或菜单栏输入中定义的系统设定值。菜单管理器还可以使用和 mctRGB2 域中定义的同样彩色绘制一个菜单命令项目的黑白图标（使用 'cicn' 资源可以为菜单命令项目提供一个彩色图标）
- mctRGB3指定菜单命令项目的等效命令组合键彩色。如果一个菜单命令项目没有菜单命令项目输入，菜单管理器就使用在菜单标题输入或菜单栏输入中定义的系统设定值
- mctRGB4指定菜单的背景色。如果菜单色彩信息表没有该项目所在菜单的菜单标题输入，或没有菜单栏输入，菜单管理器就会使用该域的值作为菜单的背景色。如果已经有了一个菜单标题输入，那么菜单管理器将用菜单标题输入域 mctRGB4 定义的值（或菜单栏输入域 mctRGB2 定义的值）代替菜单命令项目输入 mctRGB4 中的值

您可以使用 GetMCInfo 函数来得到应用程序的菜单色彩信息表的一个拷贝，使用 SetMCEntries 过程来设定应用程序菜单色彩信息表的输入，或者您可以为您的菜单提供定义色彩输入的 'mctb' 资源。

GetMenu、GetNewMBar 和 ClearMenuBar 例程也可以修改菜单色彩信息表中的输入。GetMenu 函数可以用等于 menuID 参数中值的资源标识符查找一个 'mctb' 资源。如果找到一个，它便将输入添加到菜单色彩信息表中。

在 GetNewMBar 函数创建一个新的菜单列表时，它可以建立一个新的菜单色彩信息表。如果您需要存储当前菜单色彩信息表，则应在调用 GetNewMBar 前调用 GetMCInfo。

ClearMenuBar 过程可以重新初始化当前菜单列表和菜单色彩信息表。

菜单管理器例程

菜单管理器包括用来创建菜单、改变菜单命令项目的特征以及处理用户对菜单命令选择的例程。菜单管理器还提供许多例程来完成必要的工作，包括向菜单添加项目或从菜单删除项目，统计菜单中的项目数，取得菜单的菜单记录间接指针，部署菜单，计算菜单的尺寸，醒目显示菜单栏，以及管理应用程序的菜单色彩信息表中的输入。

一些菜单管理器例程的名称可以有多种拼写方法，您开发环境支持的界面文件使用其中的任意一种方法来存取该例程。例如，GetMenuHandle 也可用 GetMHandle 得到。表 3-8 显示了例程的原来名称和它新的等效名称之间的对照情况。

表 3-8　菜单管理器例程新旧名称对照

新名称	旧名称
AppendResMenu	AddResMenu
DeleteMCEntries	DelMCEntries
DeleteMenuItem	DelMenuItem
DisposeMCInfo	DispMCInfo
GetMenuHandle	GetMHandle
GetMenuItemText	GetItem
InsertMenuItem	InsMenuItem
SetMenuItemText	SetItem

初始化菜单管理器

您可以使用 InitMenu 过程来初始化菜单管理器。

您可以使用 InitProcMenu 过程来设定当前菜单列表，在必要时，还可以使用一个自定菜单栏定义函数。

InitMenu

InitMenu 过程在您的应用程序堆中为应用程序的当前菜单列表分配存储器空间。为了初始化菜单管理器和应用程序的当前菜单列表，您只需调用 InitMenu 一次。

```
PROCEDURE InitMenus;
```

说明

InitMenu 过程可以创建没有菜单、子菜单或弹出式菜单的当前菜单列表。InitMenu 还可以创建应用程序的菜单色彩信息表。另外，在分配了菜单色彩信息表后，InitMenu 还可以使用资源标识符 0 查找一个 'mctb' 资源。如果您希望在应用程序的菜单栏和菜单中使用系统设定色彩之外的其它彩色，您可以提供一个资源标识符为 0 的 'mctb' 资源作为应用程序的一个资源。若 InsertMenu 找到并成功地装入了一个 'mctb' 资源，那么它就会将包含在该资源中的信息添加到菜单色彩信息表中（使用 SetMCEntries）。

InsertMenu 过程还可以绘制一个空菜单栏。

特殊考虑

您的应用程序在初始化菜单管理器之前，必须初始化 QuickDraw、字体管理器和窗口管理器（使用 InitGraf、InitFonts 和 InitWindows 过程）。

亦请参阅

若要为您的应用程序的菜单栏设置菜单，可以使用 GetNewMBar 和 SetMenuBar，您可以参阅 3-111 页和 3-112 页中有关这两个函数的说明。您也可以使用 3-108 页中介绍的 InsertMenu 过程为当前菜单列表添加菜单。

若要从当前菜单列表中取消所有的菜单，可以使用 3-110 页中介绍的 ClearMenuBar 过程。

如果您的应用程序希望使用它自己的菜单栏定义函数，可以使用 InitProcMenu 过程将当前菜单列表的 mbResID 域设定为自定 'MBDF' 资源的资源标识符。

有关 'mctb' 资源的说明请参阅 3-155 页中的“菜单色彩信息表资源”。

有关 InitWindows 过程的说明请参阅本书的“窗口管理器”一章。有关 InitGraf 和 InitFonts 的说明，请参阅《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*) 和《Inside Macintosh 文本》(*Inside Macintosh: Text*)。

InitProcMenu

苹果公司建议您使用标准的菜单栏定义函数。但是，如果您的应用程序具有它自己的菜单栏定义函数，应使用 InitProcMenu 过程将当前菜单列表的 mbResID 域设定为自定 'MBDF' 资源的资源标识符。

```
PROCEDURE InitProcMenu (resID: Integer);
```

`resID` 您的应用程序的菜单栏定义函数的资源标识符在该参数的高位字节 13 位；差异在低位字节 3 位。您必须使用一个大于 $ 100 的资源标识符。

对 'MBDF' 类型的资源，苹果公司保留 $ 000- $ 100 之间的资源标识符以备自用。

说明

如果当前菜单列表在上一次调用 InitMenus 时没有创建，InitProcMenu 过程就创建它。InitProcMenu 过程可以储存您在当前菜单列表 mbResID 域指定的资源标识符。菜单管理程序可以使用在该域引用的菜单栏定义函数来绘制菜单栏并对菜单执行基本操作。

特殊考虑

在您的应用程序下次调用 InitMenus 之前，它的菜单栏定义函数的资源标识符始终保留在当前菜单列表中；InitMenus 可以使用标准菜单栏定义函数初始化 mbResID 域。这可能会影响应用程序，例如控制其它可能调用 InitMenus 的应用程序的开发环境。

亦请参阅

请参阅 3-103 页中 InitMenus 的说明；如果您的应用程序使用标准的菜单栏定义函数，那么您就应该使用 InitMenus。

创建菜单

虽然您常常通过提供一个 'MBAR' 资源并使用 GetNewMBar 函数一次创建您菜单栏中的所有菜单，但您也可以使用 NewMenu 或 GetMenu 函数来创建一个下拉式菜单。有关创建菜单栏的信息，请参阅 3-112 页中的“存取菜单栏”。通常，您可以使用 NewMenu 或 GetMenu 函数来创建子菜单或弹出式菜单。

NewMenu函数可以创建一个具有指定标题的菜单，给它指派指定的菜单标识符，并为菜单创建一个菜单记录。应使用 AppendMenu、InsertMenu、AppendResMenu 或 InsertResMenu 为 NewMenu 创建的菜单添加项目。

GetMenu函数创建的菜单具有在指定的 'MENU' 资源中定义的标题、项目和特征。

NewMenu 和 GetMenu 都在您的应用程序堆中为菜单记录分配空间，并为菜单新创建的菜单记录返回一个间接指针。

如果需要向当前菜单列表添加 NewMenu 或 GetMenu 创建的菜单，可以使用 InsertMenu 过程。如果需要用任何新菜单标题更新菜单栏，可以使用 DrawMenuBar。

NewMenu

您可以使用NewMenu函数来创建一个具有指定标题和菜单标识符的空菜单。在大多数情况下，您应该在资源中储存菜单的信息（例如它们的标题、项目和特征）；应使用 GetMenu 或 GetNewMBar 函数从资源定义中创建菜单。

```
FUNCTION NewMenu (menuID: Integer; menuTitle: Str255): MenuHandle;
```

`menuID` 菜单的菜单标识符。(注意，它不是 'MENU' 资源的资源标识符)。菜单标识符是标识菜单的一个数。对于您的应用程序菜单，应使用正数作为菜单标识符。对于桌面附件(除了桌面附件的子菜单)则应使用负数作为菜单标识符。子菜单的菜单标识符必须在1到255之间。对于一个应用程序的子菜单，它的菜单标识符应该在1到235之间；对于桌面附件的子菜单，它的菜单标识符应该在 236 到 255 之间。苹果公司保留菜单标识符 0。

`menuTitle` 新菜单的标题。注意，在大多数情况下您应该在资源中储存标题，以便您的菜单标题更容易本地化。

说明

NewMenu 函数可以创建一个具有指定标题的菜单，给它指派指定的菜单标识符，为菜单创建一个菜单记录，并为菜单记录返回一个间接指针。它可以设置菜单记录，以便使用标准菜单定义过程 若标准菜单定义过程尚未在存储器中，NewMenu 将它读入)。NewMenu 函数不能将新创建的菜单插入当前菜单列表。

在使用 NewMenu 创建了一个菜单后，就应该使用 AppendMenu、InsertMenuItem、AppendResMenu 或 InsertResMenu 将菜单命令项目添加到菜单中。如果需要将 NewMenu 创建的菜单添加到当前菜单列表中，可以使用 InsertMenu 过程。如果需要用任何新的菜单标题更新菜单栏，则可以使用 DrawMenuBar 过程。

特殊考虑

若要释放和您用 NewMenu 创建的菜单有关的存储器，应首先调用 DeleteMenu 从当前菜单列表中删除该菜单，并取消该菜单在应用程序的菜单色彩信息表中的所有输入项，然后调用 DisposeMenu 来处理该菜单的菜单记录。在处理了一个菜单后，应使用 DrawMenuBar 来更新菜单栏。

若 NewMenu 不能创建菜单记录，它会返回 NIL 作为它的函数结果。

亦请参阅

有关如何将项目添加到菜单中的信息，请参阅: 3-124 页中对 AppendMenu 的说明，3-126 页中对 InsertMenuItem 的说明，3-128 页中对 AppendResMenu 的说明，3-129 页中对 InsertResMenu 的说明。有关 InsertMenu 的信息，请参阅 3-108 页。若要处理一个菜单，请参阅 3-109 页中对 DeleteMenu 的说明和 3-140 页中对 DisposeMenu 的说明。

GetMenu

可以通过调用 GetMenu 函数来创建一个菜单，它的标题、项目和其它特征是在具有指定资源标识符的 'MENU' 资源中定义的。通常，您只有在创建子菜单时才使用这个函数；您可以使用 GetNewMBar 函数一次创建完成所有的下拉式菜单，您可以使用标准的推出控制部件定义函数创建弹出式菜单。

```
FUNCTION GetMenu (resourceID : Integer) : MenuHandle ;
```

resourceID　定义菜单特征的 'MENU' 资源的资源标识符。通常，您需要一个菜单的资源标识符，该资源标识符和您在菜单资源中的菜单标识符相同)。

说明

GetMenu 函数可以根据指定的菜单资源创建一个菜单，它还可以为菜单创建菜单记录。如果菜单定义过程 在菜单资源中指定）尚未在存储器中，GetMenu会将它读入存储器，关在菜单记录中储存一个菜单定义过程的间接指针。GetMenu函数不能把新创建的菜单插入当前菜单列表。

读取 'MENU' 资源后，GetMenu 函数将查找和 'MENU' 资源有同样资源标识符的 'mctb' 资源。如果 GetMenu 找到了这个 'mctb' 资源，它便使用其中的信息将该菜单的入口地址添加到应用程序的菜单色彩信息表中。GetMenu 函数可以使用 SetMCEntries 将 'mctb' 资源定义的入口地址添加到应用程序的菜单色彩信息表中。如果 GetMenu 函数没有找到这个 'mctb' 资源，它便使用应用程序菜单栏中的菜单色彩信息的菜单栏输入中定义的系统设定色彩，若该信息不存在，便使用菜单的标准色彩。

GetMenu函数为该菜单的菜单记录返回一个间接指针。您可以在大多数菜单管理器例程中使用返回的菜单间接指针来引用这个菜单。若 GetMenu 未能在资源文件中读取菜单或菜单定义过程，GetMenu 返回 NIL。

在使用 GetMenu 创建一个菜单后，您可以在必要时使用 AppendMenu、InsertMenu-Item、AppendResMenu 或 InsertResMenu 将更多的菜单命令项目添加到菜单中。

若要将 GetMenu 创建的一个菜单添加到菜单列表中，应使用 InsertMenu 过程。若要用任何新的菜单标题更新菜单栏，则应使用 DrawMenuBar 过程。

将菜单的定义储存在资源中 特别是菜单标题和菜单命令项目），以便使您的应用程序易于本地化。

▲ **警告**

一个资源中的菜单必须是不可清除的菜单。

特殊考虑

若要释放 GetMenu 从资源文件中读取的菜单所占据的存储器，应首先调用 DeleteMenu 从菜单列表中删除该菜单，并删除该菜单在应用程序的菜单色彩信息表中的所有菜单标题输入或菜单命令项目输入，然后调用资源管理器过程 ReleaseResource 来处理该菜单的菜单记录。另外，使用 DrawMenuBar 来更新菜单栏。

▲ **警告**

对于一个特定的菜单只能调用 GetMenu 一次。如果您需要当前菜单列表中的一个菜单的间接指针，则应使用 GetMenuHandle 或资源管理器函数 GetResource.

亦请参阅

有关 'MENU' 资源的说明，请参阅 3-151 页中的“菜单资源”；有关 Rez 格式的一个 'MENU' 资源的示例，请参阅 3-48 页中的程序清单 3-2。有关 'mctb' 资源的信息，请参阅 3-155 页中的“菜单色彩信息表资源”。

有关如何将项目添加到一个菜单中的细节，请参阅 3-124 页中对 AppendMenu、3-126 页中对 InsertMenuItem、3-128 页中对 AppendResMenu，及 3-129 页中对 InsertResMenu 的说明。若要删除一个菜单，请参阅 3-109 页中对 DeleteMenu 的说明。如果希望更新菜单栏，可以使用 3-113 页中介绍的 DrawMenuBar 过程。

向或从当前菜单列表中添加或删除菜单

在用 NewMenu 或 GetMenu 创建一个菜单后，应使用 InsertMenu 过程将菜单插入到当前菜单列表中。可以通过 DeleteMenu 过程从当前菜单列表中删除一个；或使用 ClearMenuBar 过程从当前菜单列表中清除所有的菜单。

InsertMenu

InsertMenu 功能为在当前菜单列表中插入一个菜单。

```
PROCEDURE InsertMenu (theMenu: MenuHandle; beforeID: Integer);
```

`theMenu` 菜单的菜单记录间接指针。NewMenu 和 GetMenu 函数为一个菜单记录返回一个间接指针，您可以通过这个参数使用它。

`beforeID` 一个可以用来表明菜单应插入当前菜单列表中什么位置的数字。InsertMenu 将菜单插入当前菜单列表中某个菜单之前，该菜单的菜单标识符等于在 beforeID 参数中指定的数字。若 beforeID 中的参数为 0（或它不是菜单列表中任何菜单的 ID），InsertMenu 将新菜单插入在所有其它菜单之后（除了在帮助、键盘输入和应用程序菜单之前）。若菜单已经在当前菜单列表中，或菜单列表已经排满，InsertMenu 则不做任何事情。

您可以通过将 beforeID 参数指定为 -1 来将子菜单插入当前菜单列表。菜单列表子菜单部分的子菜单在当前并不一定要附属于某个嵌套式菜单的菜单命令项目；您可以将子菜单储存在菜单列表中，以后需要时再将子菜单指定给一个菜单命令项目。MenuKey 函数能够扫描菜单列表中的所有菜单，搜索等效命令组合键，包括不附属于任何菜单命令项目的子菜单。您不应该对当前菜单列表中而又不附属于一个菜单命令项目的子菜单定义等效命令组合键。

您也可以为beforeID参数指定 -1，将一个弹出式菜单插入当前菜单列表中。但是，若您使用标准的弹出控制部件定义函数，弹出控制部件就会根据弹出控制部件的需要，自动将菜单插入当前菜单列表中。

说明

InsertMenu过程可以把指定的菜单插入当前菜单列表中。若希望更新菜单栏以反映新的菜单，可以调用DrawMenuBar。

亦请参阅

有关如何更新菜单栏的细节说明，请参阅 3-113 页中的说明。

DeleteMenu

使用DeleteMenu过程从当前菜单列表中删除一个现有的菜单。

```
PROCEDURE DeleteMenu (menuID: Integer);
```

menuID　可以从当前菜单列表中删除菜单的菜单标识符。如果菜单列表中没有指定菜单标识符的菜单，DeleteMenu将不做任何事情。

说明

DeleteMenu过程可以从当前菜单列表中删除指定的菜单，它还能够从应用程序的菜单色彩信息表中取消该菜单的所有彩色信息。DeleteMenu不释放菜单记录占据的存储器。若要释放菜单的数据结构占据的存储器，对NewMenu创建的菜单，应使用DisposeMenu过程；对GetMenu创建的菜单或用GetNewMBar读入资源的菜单，应使用资源管理器过程ReleaseResource。

DeleteMenu过程首先在当前菜单列表的子菜单部分核查符合指定 ID 的菜单标识符。若找到了这样的菜单，它就删除该菜单并返回。若DeleteMenu在子菜单部分未找到这样的菜单，它将再次核查当前菜单列表的有效部分。这就允许桌面附件删除一个子菜单，而不用删除一个其菜单标识符可能和桌面附件定义的菜单标识符冲突的应用程序菜单。

在删除一个菜单后，应使用DrawMenuBar 来更新菜单栏，以便反映当前菜单列表的更改情况。

亦请参阅

有关如何使用 DisposeMenu 处理菜单数据结构的细节，请参阅 3-140 页中的“处理菜单”。有关 ReleaseResource 过程的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书的“资源管理器”一章。

ClearMenuBar

使用 ClearMenuBar 过程从当前菜单列表中删除所有的菜单。

```
PROCEDURE ClearMenuBar;
```

说明

ClearMenuBar过程可以从当前菜单列表中删除所有的菜单，并能够从应用程序的菜单色彩信息表中删除所有的色彩信息。ClearMenuBar不释放菜单的菜单记录或菜单色彩信息表所占据的任何存储器。若要释放菜单的数据结构占据的存储器，对 NewMenu 创建的每个菜单，可以使用 DisposeMenu 过程；对 GetMenu 创建的每个菜单或用 GetNewMBar 读入资源的每个菜单，可以使用 ReleaseResource 过程。

在从当前菜单列表中删除了所有菜单后，应使用 DrawMenuBar 来更新菜单栏的外观。

亦请参阅

若要更新一个应用程序的菜单栏，请参阅 3-113 页中对 DrawMenuBar 的说明。有关 ReleaseResource 过程的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书中“资源管理器”一章。

从 'MBAR' 资源中获取菜单栏说明信息

通常您可以通过执行下列步骤来创建应用程序的菜单栏:

- 在一个 'MBAR' 资源中定义菜单的次序和资源标识符
- 在 'MENU' 资源中定义菜单
- 用 GetNewMBar 函数读入这些说明
- 将当前菜单列表设定为 GetNewMBar 返回的菜单列表
- 用 DrawMenuBar 更新菜单栏

GetNewMBar

使用 GetNewMBar 函数从一个 'MBAR' 资源读入菜单栏的定义。

```
FUNCTION GetNewMBar (menuBarID: Integer): Handle;
```

menuBarID　'MBAR' 资源的资源标识符，它为一个菜单栏指定了菜单。

说明

GetNewMBar 函数可以从一个 'MBAR' 资源读入一个菜单栏的定义及其菜单。'MBAR' 资源标示菜单栏中菜单的次序。另外，它还为每个菜单指定了菜单资源标识符。GetNewMBar 函数从 'MBAR' 资源读入每个菜单，其资源标识符在 'MBAR' 资源中指定。

GetNewMBar 函数可以为 'MBAR' 资源定义的菜单栏创建一个菜单列表，并为菜单列表返回一个间接指针。(若资源尚未在存储器中，GetNewMBar 便将它读入存储器。) 若 GetNewMBar 不能读取资源，它便返回 NIL。GetNewMBar 可以使用 GetMenu 读入单个的菜单。

当读取某个 'MBAR' 资源后，可以使用 SetMenuBar 令 GetNewMBar 创建的菜单列表成为当前菜单列表。然后，使用 DrawMenuBar 来更新菜单栏。

若要释放一个菜单列表中菜单的数据结构占据的存储器，对于 NewMenu 创建的每个菜单，可以使用 DisposeMenu；对于 GetMenu 创建的每个菜单或用 GetNewMBar 读入资源的每个菜单，则应使用资源管理器过程 ReleaseResource。若要释放菜单列表占据的存储器，可以使用菜单管理器过程 DisposeHandle。

特殊考虑

GetNewMBar 函数首先存储当前菜单列表，然后清除当前菜单列表以及应用程序的菜单色彩信息表。随后，它会创建一个新的菜单列表。在为新的菜单列表返回间接指针之前，GetNewMBar 函数会将当前菜单列表还原成原来存储的菜单列表，但是 GetNewMBar 不还原原来的菜单色彩信息表。若要存储并随后还原应用程序的当前菜单色彩信息表，应在调用 GetNewMBar 前调用 GetMCInfo 函数，并在此后再次调用 SetMCInfo 过程。

尽管您只为 GetNewMBar 函数提供 'MBAR' 资源的资源标识符，但是您的应用程序常常需要使用每个菜单 'MENU' 资源中定义的菜单标识符。大多数菜单管理器的例程或是要求一个菜单标识符，或是一个菜单记录间接指针，来对指定的菜单执行操作。对于当前菜单列表中的菜单，您可以使用 GetMenuHandle 函数来获取具有给定菜单标识符的菜单的菜单记录间接指针。

亦请参阅

有关 'MENU' 资源的说明，请参阅 3-151 页中的“菜单资源”；有关一个 Rez 格式的示例 'MENU' 资源，请参阅 3-48 页中的程序清单 3-2。有关 'MBAR' 资源的说明，请参阅 3-155 页中的“菜单栏资源”；有关一个 Rez 格式的示例 'MBAR' 资源，请参阅 3-49 页中的程序清单 3-4。有关 ‘mctb’ 资源的信息，请参阅 3-155 页中的“菜单色彩信息表资源”。有关资源管理器的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*)。

获取和设定菜单栏

您可以使用 GetMenuBar 函数为当前菜单列表的一个拷贝获取一个间接指针。使用 SetMenuBar 过程将当前菜单栏设定为 GetMenuBar 或 GetNewMBar 原来返回的菜单列表。您可以用 GetMBarHeight 函数获取菜单栏的高度。

GetMenuBar

使用 GetMenuBar 函数为当前菜单列表的一个拷贝取得一个间接指针。

```
FUNCTION GetMenuBar: Handle;
```

说明

GetMenuBar 函数可以创建当前菜单列表的一个拷贝，并能够为该拷贝返回一个间接指针。您可以存储返回的菜单列表，然后向当前菜单列表添加或从当前菜单列表删除菜单（使用 InsertMenu、DeleteMenu 或 ClearMenuBar）。随后可以使用 SetMenuBar 还原存储的菜单列表。

若要释放存储的菜单列表占据的存储器，应使用菜单管理器的 DisposeHandle 过程。

▲ 警告

GetMenuBar 不复制菜单记录，只复制菜单列表（它包含菜单记录的间接指针）。如果您希望在以后还原菜单列表，那么就请不要在菜单列表中处理任何菜单。

SetMenuBar

使用 SetMenuBar 过程将当前菜单列表设定到一个指定的菜单列表。

```
PROCEDURE SetMenuBar (menuList: Handle);
```

`menuList` 一个菜单列表的间接指针，它能够为一个菜单栏指定菜单。您应该指定一个 GetMenuBar 或 GetNewMBar 返回的间接指针。

说明

SetMenuBar 过程可以将给定的菜单列表拷贝到当前菜单列表。SetMenuBar 和 GetMenuBar 一样，不复制菜单记录，只复制菜单列表（它包含菜单记录的间接指针)。

您可以使用SetMenuBar来还原您原来用GetMenuBar存储的菜单列表，或者将当前菜单列表设定为 GetNewMBar 创建的菜单列表。

SetMenuBar过程只设定当前菜单列表；若要根据新的菜单列表更新菜单栏，应使用 DrawMenuBar 过程。

GetMBarHeight

如果您需要确定菜单栏的当前高度，可以使用GetMBarHeight 函数。当罗马语系系统是当前系统语系时，菜单栏高度是20像素。若当前系统语系不是罗马语系，那么为了适应当前系统字体，菜单栏的高可能会超过20像素。

```
FUNCTION GetMBarHeight: Integer;
```

说明

GetMBarHeight 函数以像素为单位返回菜单栏的当前高度。

绘制菜单栏

只要您的应用程序向当前菜单列表添加或从当前菜单列表删除菜单，您应该调用 DrawMenuBar过程更新菜单栏中的菜单标题。若您改变了菜单的有效状态，您应该调用 DrawMenuBar 相应地更新菜单标题。另一种方法是，您可以不使用 DrawMenuBar 过程，而使用 InvalMenuBar 过程令菜单栏作废；这可以使事件管理程序重新绘制菜单栏，作为它正常处理更新事件过程的一部分。

DrawMenuBar

使用 DrawMenuBar 过程并依据当前菜单列表绘制菜单栏。

```
PROCEDURE DrawMenuBar;
```

说明

DrawMenuBar 过程可以根据当前菜单列表绘制或重新绘制菜单栏。在下列几种情况下，您必须调用 DrawMenuBar 来更新菜单栏:

(1) 使用InsertMenu或DeleteMenu向当前菜单列表添加或从当前菜单列表删除菜单后;
(2) 使用 SetMenuBar 设定当前菜单列表后;
(3) 更改一个菜单的有效状态后;
(4) 在使用任何更改当前菜单列表的例程后。

InvalMenuBar

使用 InvalMenuBar 过程令菜单栏作废。

```
PROCEDURE InvalMenuBar;
```

说明

InvalMenuBar过程能够为菜单栏设定已被更改和需要更新两种标记。当事件管理器扫描更新区域搜索需要进行更新的区域时，事件管理器也要通过核查确定菜单栏是否需要更新（因为调用了 InvalMenuBar）。如果菜单栏需要更新，那么事件管理器便会调用 DrawMenuBar 过程来绘制菜单栏。

您可以使用 InvalMenuBar 来代替 DrawMenuBar，以便减少菜单栏的闪烁时间。例如，若您有几个应用程序定义的例程，它们能够改变菜单的有效状态，每个例程都需调用 DrawMenuBar ，您可以调用 InvalMenuBar 来代替调用 DrawMenuBar 过程。这样，只需重新绘制菜单栏一次，而不是快速相继绘制多次。若您需要立即更改菜单栏，则应使用 DrawMenuBar。若您希望在您事件循环的每一周内，最多重新绘制菜单栏一次，则应使用 InvalMenuBar。InvalMenuBar 过程只有在 System 7 中才能使用。

响应用户选择的菜单命令

当用户按下鼠标按钮，同时光标在菜单栏上时，您的应用程序应该调用 MenuSelect 函数，允许用户在菜单栏中选择一个菜单。如果用户按下鼠标按钮，同时光标在未使用标准弹出控制部件定义函数的弹出式菜单上，那么您的应用程序就应该调用 PopUpMenuSelect 函数，允许用户在弹出式菜单中进行选择。您还应该允许通过用户键入等效命令组合键来选择一个菜单命令。当用户在键盘上按下一个键时，您的应用程序应该确定是否同时按下了 Command 键。如果同时按下了 Command 键，您的应用程序应该调用 MenuKey 函数将这个组合键映射到相应的等效命令组合键。

如果用户选择了一个项目，那么 MenuSelect 和 MenuKey 函数都将醒目显示包含所选项目的菜单标题，并向您的应用程序报告用户的选择。您的应用程序执行相应的命令，并在执行完毕后调用 HiliteMenu 过程取消菜单标题的醒目显示，这就向用户表明他的命令已经完成了。

如果用户释放了鼠标按钮，同时光标在一个无效的命令项目上，或者用户键入了一个无效项目的等效命令组合键，MenuSelect 和 MenuKey 不报告菜单标识符或项目的菜单命令项目。如果需要确定用户是否选择了一个无效的命令项目（例如，您的应用程序能够为用户提供帮助或向用户解释为什么命令是停用的），您可以使用 MenuChoice 函数返回无效菜单命令的菜单标识符和菜单命令项目。

您的应用程序在调用 MenuSelect 或 MenuKey 前，应该调整它的菜单。例如，适当地启用或停用菜单命令项目，并为显示属性的项目添加相应的选择标记或破折号。

MenuSelect

使用 MenuSelect 函数，允许用户在应用程序菜单栏的菜单中选择一个菜单。

```
FUNCTION MenuSelect (startPt: Point): LongInt;
```

`startPt` 代表按下鼠标时光标所在位置（以全局坐标表示）。

说明

当用户按下鼠标按钮，同时光标在菜单栏上时，您的应用程序会收到一个鼠标按下事件。如果需要处理菜单栏中的鼠标按下事件，可以将鼠标按下事件发生时的光标位置作为 startPt 参数传送给 MenuSelect。当用户将光标移过菜单栏的菜单标题时，MenuSelect 函数显示并删除菜单，它还在用户释放鼠标按钮前处理所有人机交互。

当用户拖拽光标通过菜单栏时，MenuSelect函数醒目显示光标当前经过的菜单标题，并显示该菜单挂有的所有项目。若用户移动光标移动另一个菜单上，MenuSelect 函数则删除原来的菜单并取消菜单标题的醒目显示。

当用户拖拽光标通过菜单命令项目时，MenuSelect 函数醒目显示菜单命令项目或取消菜单命令项目的醒目显示。当光标通过一个有效的菜单命令项目时，MenuSelect 函数醒目显示它；当用户将光标移至另一个菜单命令项目时，MenuSelect 取消它的醒目显示。MenuSelect 将不醒目显示无效的菜单命令项目。

如果用户选择一个启用的菜单命令项目（包括子菜单的所有项目），MenuSelect 函数便会返回一个值作为它的函数结果，指示用户选择了哪一个菜单和菜单命令项目。函数结果的高位字包含菜单的菜单标识符，低位字包含了用户选择的菜单命令项目的项目编号。MenuSelect 函数能够保持菜单标题的醒目显示；在执行了选择的任务后，您的应用程序应该使用 HiliteMenu 过程取消菜单标题的醒目显示。

如果用户从一个子菜单中选择一个项目，MenuSelect 将会在其函数结果的高位字返回子菜单的菜单标识符，在低位字返回用户选择的项目。菜单栏中如果有原来用户为了开始进入子菜单而显示的菜单，MenuSelect 函数也将醒目显示它的标题。在执行了选择的任务后，您的应用程序应该取消菜单标题的醒目显示。

如果用户释放鼠标按钮，同时光标在菜单栏的一个无效的项目上，那么在菜单栏上或在任何项目之外，MenuSelect 函数会在其函数结果的高位字返回 0，低位字未定义。若您的应用程序必须找到无效项目的项目编号，它可以调用 MenuChoice 来返回菜单标识符和菜单命令项目。

如果用户在桌面附件插入应用程序菜单列表中的一个菜单中选择了有效的项目，MenuSelect 将会使用 SystemMenu 过程来处理这种情况，并在高位字给应用程序返回 0。

特殊考虑

当 MenuSelect 函数拉下一个菜单时，它将菜单后面的位在您的应用程序堆中作为可重定位的对象储存。

汇编语言信息

InitMenu 和 InitProcMenu 过程可以将 MenuHook 和 MBarHook 全局变量初始化为 0。您也可以选择在这些全局变量中储存 MenuSelect 调用的例程的地址。MenuHook 全局变量包含鼠标按钮按下时 MenuSelect 重复调用的例程的地址（如果有的话）。MenuSelect 不向这个例程传送任何参数。

MBarHook 全局变量包含 MenuSelect 在菜单标题被醒目显示，菜单矩形已经计算之后，但菜单绘制之前调用的例程的地址。菜单矩形是一个矩形框（以全局坐标表示），菜单将在其中绘制。MenuSelect 将一个指针传送到堆栈的菜单矩形。若您在 MBarHook 全局变量中提供了例程的地址，正常情况下，它应该在 D0 寄存器返回 0，表明 MenuSelect 应该继续；返回 1 将使得 MenuSelect 取消它的操作，并立即返回应用程序。

MenuSelect 函数可以使用全局变量 MBarEnable 来确定当前菜单栏的所有菜单是属于桌面附件还是属于一个应用程序。如果 MBarEnable 全局变量不是 0，那么当前菜单栏中的所有菜单属于一个桌面附件。如果 MBarEnable 全局变量是 0，那么当前菜单栏中的所有菜单便属于一个应用程序。如果您在编写一个桌面附件，您可能需要将 MBarEnable 全局变量设定为非 0；如果您在编写一个应用程序，您就不应该更改 MBarEnable 全局变量的值。

全局变量 TheMenu 包含了菜单栏中当前醒目显示的菜单的资源标识符。若用户从子菜单中选择一个项目，TheMenu 就会包含子菜单的菜单标识符，而不是子菜单所附属的菜单的菜单标识符。

亦请参阅

有关在调用 MenuSelect 前，调整您的应用程序菜单的信息，请参阅 3-73 页中的“调整一个应用程序的菜单”。有关如何取消菜单醒目显示的细节，请参阅 3-119 页中对 HiliteMenu 过程的说明。有关如何确定用户是否选择了一个停用项目的信息，请参阅 3-118 中对 MenuChoice 函数的说明。

MenuKey

如果用户在按下Command键的同时，按下了另一个键，则应调用MenuKey函数来确定组合键是否映射到当前菜单列表中一个菜单之菜单命令项目的等效命令组合键。

```
FUNCTION MenuKey (ch: Char): LongInt;
```

ch　　1字节字符，该字符代表用户和Command键一同按下的键。

说明

MenuKey函数可以将给定的字符映射到菜单和具有该等效命令组合键的菜单命令项目。MenuKey函数返回一个值作为函数结果，该值表示菜单标识符和具有相应于给定字符的等效命令组合键的菜单命令项目。

MenuKey函数不区分大写和小写字母。它接受传送给它的一个字符，并调用UpperText过程（它提供了字符的可本地化的大写转换）。这样，MenuKey在比较键盘事件和等效命令组合键时，会将所有小写字母转换成大写字母。这允许用户去请求一个等效命令组合键命令，例如同时按下Command键和“c”或“C”键请求“复制”命令。为了在应用程序间保持一致，应该在菜单中将命令的等效命令组合键定义为大写字母。

如果给定的字符映射到当前菜单列表中一个有效的菜单命令项目，那么MenuKey将醒目显示所选菜单的菜单标题，并在其函数结果的高位字返回菜单标识符，在函数结果的低位字返回所选的菜单命令项目。在执行了所选的任务后，您的应用程序应该使用HiliteMenu过程取消菜单标题的醒目显示。

若给定的字符未映射到当前菜单列表中一个有效的菜单命令项目，MenuKey在它的高位字返回0，它的低位字未定义。

若给定的字符映射到一个桌面附件插入应用程序菜单列表中的菜单的菜单命令项目，MenuSelect便会使用SystemMenu过程来处理这个情况，并在高位字向您的应用程序返回0。

您不应该用相同的等效命令组合键定义菜单命令项目。MenuKey从右至左扫描菜单，从上到下扫描项目。如果您已经用相同的等效命令组合键定义了一个以上的菜单命令项目，那么MenuKey将返回它首先找到的那个项目。

MenuKey函数首先在当前菜单列表的常规部分搜索具有匹配给定键的等效命令组合键的菜单命令项目。若它在那里没有找到，就搜索当前菜单列表的子菜单部分。若给定键映射到一个子菜单的菜单命令项目，MenuKey醒目显示菜单栏中用户将正常地拉下以进入子菜单的菜单标题。您的应用程序应该执行要求的命令，然后取消菜单标题的醒目显示。

您不应该指派Command—Shift—数字键序列作为一个菜单命令项目的等效命令组合键；Command—Shift—数字键序列将被保留，以便用于‘FKEY’资源。Command—Shift—数字键序列将不会被返回您的应用程序，而是由事件管理器处理。事件管理器将会请求一个带有资源标识符的'FKEY'资源，该资源标识符对应着激活'FKEY'资源的号码。

苹果公司保留“Command”＋键代码＄1B（控制—［）到＄1F（控制—_）来表示等效命令组合键以外的意义。如果您在ch参数中指定了任何这样的值，MenuKey将忽略这些字符代码并返回函数结果0。您的应用程序不应该自己使用这些字符代码。

全局变量 TheMenu 包含菜单栏中当前醒目显示的菜单。如果用户从该子菜单中选择了一个项目，TheMenu将包含子菜单的菜单标识符，而不是子菜单所属的菜单的菜单标符。

▲ 警告

请不要定义“循环”嵌套式菜单，即一个带有子菜单之子菜单的嵌套式菜单，后者的子菜单是菜单链中一个较高的嵌套式菜单。若MenuKey检测到一个循环嵌套式菜单，它将产生一个错误号码为 86 的系统错误。

亦请参阅

如果要取消醒目显示，可以使用 3-119 页中说明的HiliteMenu过程。如果希望提供等效命令组合键之外的等效命令组合键，请参阅《Inside Macintosh : Text》一书中关 'KCHR' 的论述。

MenuChoice

如果您的应用程序希望找到用户试图选择的无效菜单命令项目的项目编号，您可以使用MenuChoice函数来返回所选的菜单命令项目号。

```
FUNCTION MenuChoice: LongInt;
```

说明

如果用户选择了一个停用菜单命令项目，MenuChoice函数将返回一个值，该值可以指示用户选择的是那个菜单和菜单命令项目。函数结果的高位字包含菜单的菜单标识符，低位字包含了用户所选的菜单命令项目的项目编号。

如果当光标在菜单栏或在菜单之外时，鼠标按钮释放，那么MenuChoice函数将返回0作为其函数结果的低位字。

特殊考虑

无论什么时候只要显示菜单，菜单管理器就会更新全局变量MenuDisable。当用户将光标移过每个项目时，菜单管理器调用菜单的菜单定义过程来更新MenuDisable全局变量，以反映当前菜单标识符和菜单命令项目。标准的菜单定义过程能够适当地更新全局变量MenuDisable。如果您的应用程序使用自己的菜单定义过程，那么该菜单定义过程也应该支持这个功能；如果您使用了一个不能正确更新全局变量MenuDisable的菜单定义过程，MenuChoice返回的结果便是未定义的。

HiliteMenu

您可以使用HiliteMenu过程来醒目显示菜单标题或取消菜单标题的醒目显示。例如，在执行了用户选择的菜单命令后，应使用HiliteMenu过程来取消菜单标题的醒目显示。

```
PROCEDURE HiliteMenu (menuID: Integer);
```

`menuID` 标题醒目显示的菜单的菜单标识符。若指定菜单的菜单标题已经醒目显示了，HiliteMenu不做任何事情。若菜单标识符是 0 或指定的菜单标识符不在当前菜单列表中，HiliteMenu将取消当前醒目显示的任何菜单标题（如果有的话）的醒目显示。

说明

MenuSelect和MenuKey函数醒目显示包含用户所选项目的菜单标题。在执行了选择的任务后，您的应用程序应该通过调用 HiliteMenu 取消菜单标题的醒目显示，并在menuID 参数中传送0。

HiliteMenu过程而后才能达到首先存储标题矩形后面的位，然后绘制醒目显示的标题，醒目显示一个菜单标题的目的。通过还原菜单标题后面的位，HiliteMenu可以取消一个菜单栏的醒目显示。

全局变量TheMenu包含菜单栏中当前醒目显示的菜单的标识符。如果用户从子菜单中选择一个项目，TheMenu将包含子菜单的菜单标识符，而不是子菜单所属的菜单的菜单标识符。

亦请参阅

如果希望醒目显示整个菜单栏，应使用 3-141 页中说明的FlashMenuBar过程。

PopUpMenuSelect

如果不希望使用标准的弹出控制部件定义函数而显示一个弹出式菜单，那么可以使用PopUpMenuSelect函数在屏幕的任意位置显示弹出式菜单。如果您的应用程序使用标准的弹出控制部件定义函数，那么就不需要使用 PopUpMenuSelect。

```
FUNCTION PopUpMenuSelect (menu: MenuHandle;
                          Top: Integer; Left: Integer;
                          PopUpItem: Integer)
                          : LongInt;
```

`menu` 菜单的菜单记录的间接指针。NewMenu、GetMenu和GetMenuHandle函数能够为指定菜单的菜单记录返回一个间接指针。

`Top` 弹出方框关闭时的顶部坐标。这个值应该以全局坐标给出。

`Left` 弹出方框关闭时的左侧坐标。这个值应该以全局坐标给出。

`PopUpItem` 当前项目的项目编号减 1。这个值为用户在该菜单中上一次的选择。如果用户以前没有作过什么选择，该值就为系统缺省值。

说明

PopUpMenuSelect 函数可以使用 Top 和 Left 参数指定的位置来确定显示弹出式菜单的指定项目的位置。PopUpMenuSelect 函数显示的弹出式菜单中，在 PopUpMenuSelect 参数中指定的菜单命令项目是在指定位置醒目显示的。3-34 页图 3-24 显示了弹出式标题和弹出式菜单的弹出方框。

PopUpMenuSelect函数可以醒目显示菜单命令项目或取消菜单命令项目的醒目显示，并处理所有用户对话，直至用户释放了鼠标按钮。PopUpMenuSelect 函数在其函数结果的高位字返回所选菜单的菜单标识符，在低位字返回所选菜单命令项目（注：这里可能丢了“的项目编号”）。

您的应用程序在调用PopUpMenuSelect前，负责醒目显示弹出式标题，适当设定当前菜单命令项目的标记，并在弹出方框中绘制文字和方向向下的指示符。您的应用程序还应该在调用PopUpMenuSelect前，确保弹出式菜单是在当前菜单列表中的子菜单部分。（您可以使用 InsertMenu 过程并在 beforeID 参数中指定 -1，以便将弹出式菜单插入当前菜单列表中。）

在调用 PopUpMenuSelect 后，您的应用程序会从当前菜单列表中删除弹出式菜单，也可以将它留在当前菜单列表中。

您的应用程序还负责储存菜单命令项目的当前值。在弹出式方框中绘制文字和向下的指示符，并在调用了 PopUpMenuSelect 后取消弹出式标题的醒目显示。如果您使用标准的推出控制部件定义函数，这些活动将由推出控制部件为您执行，而您的应用程序不需要调用 PopUpMenuSelect。

在执行弹出式菜单时，您应该遵循《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书中介绍的准则。例如，您应该将弹出式菜单的弹出方框定义成矩形，要求该矩形和菜单命令项目拥有相同的高度，具有 1 个像素宽度的阴影，还应该令弹出方框足够宽，以便显示当前选择的项目和方向向下的指示符。

SystemMenu

如果用户选择一个菜单中的项目，而该菜单属于在应用程序（存储器）分区中启用的一个桌面附件，那么 MenuSelect 和 MenuKey 函数就会调用 SystemMenu 过程。您的应用程序应该不需要调用 SystemMenu 过程。

```
PROCEDURE SystemMenu (menuResult: LongInt);
```

`menuResult`

指示菜单和用户选择的菜单命令项目的值。菜单标识符在高位字，菜单命令项目在低位字。属于一个桌面附件的菜单的菜单标识符 是一个负数。

说明

SystemMenu 过程通过调用桌面附件的控制例程并将 accMenu 常数传送给 csCode 数，指导桌面附件对给定的菜单命令项目执行适当的活动。桌面附件执行要求的活动后返回。请参阅《Inside Macintosh 设备驱动》(*Inside Macintosh: Device*) 一书中有关桌面附件更详细的信息。

汇编语言信息

如果您在编写一个桌面附件，您可能需要将 MBarEnable 全局变量设定为适当的值。如果 MBarEnable 全局变量不是 0，那么当前菜单栏中的所有菜单属于一个桌面附件。如果 MBarEnable 全局变量为 0，那么当前菜单栏中的所有菜单属于一个应用程序。如果您在编写一个应用程序，那么您就不应该更改 MBarEnable 全局变量的值。

SystemEdit

当用户在编辑菜单中选择一个标准编辑命令（“取消”、“剪”、“复制”、“贴”和“清除”）时，应调用SystemEdit函数来确定当前活动窗口是否属于在应用程序（存储器）分区中启用的桌面附件。如果是，SystemEdit函数指导桌面附件执行编辑命令并返回TRUE。如果当前活动窗口不属于在应用程序（存储器）分区中启用的桌面附件，那么SystemEdit就会返回FALSE，您的应用程序应该处理该命令。

```
FUNCTION SystemEdit (editCmd: Integer): Boolean;
```

editCmd　　用户选择的标准编辑命令的项目编号。

为菜单记录获取一个间接指针

大多数管理菜单的菜单管理器例程都要求您为要执行操作的菜单的菜单记录指定一个间接指针。您可以使用HMGetHelpMenuHandle函数为应用程序的帮助菜单获取一个间接指针。使用GetMenuHandle函数为应用程序的其它所有下拉式菜单或当前菜单列表中的子菜单的菜单记录获取间接指针。对于使用标准控制部件定义函数的弹出式菜单，您可以通过存取控制记录来获取菜单的间接指针。

GetMenuHandle

您可以使用GetMenuHandle函数为您的应用程序中除了帮助菜单之外的所有菜单的菜单记录获取一个间接指针。（使用HMGetHelpMenuHandle函数为您的应用程序的帮助菜单的菜单记录获取一个间接指针。）GetMenuHandle函数和GetMHandle函数是一样的。

```
FUNCTION GetMenuHandle (menuID: Integer): MenuHandle;
```

menuID　　菜单的菜单标识符。（注意，这不是资源标识符，尽管您常常指派和资源标识符相同的菜单标识符）。您在 'MENU' 资源中指派一个菜单标识符。如果您没有在 'MENU' 资源中定义您的菜单，您可以使用NewMenu指派一个菜单标识符。

说明

GetMenuHandle函数能够为具有指定菜单标识符的菜单的菜单记录返回一个间接指针。如果该菜单是在当前菜单列表中，GetMenuHandle将给该菜单的菜单记录返回一个间接指针作为它的函数结果。否则，GetMenuHandle 返回NIL作为它的函数结果。

特殊考虑

如果需要为您用弹出式控制部件定义函数创建的弹出式菜单获取一个间接指针，请不要使用 GetMenuHandle，应取消引用弹出式菜单控制记录的 cntrlData 域。

HMGetHelpMenuHandle

使用 HMGetHelpMenuHandle 函数为您的应用程序的帮助菜单获取一个间接指针。

```
FUNCTION HMGetHelpMenuHandle (VAR mh: MenuHandle): OSErr;
```

mh　　HMGetHelpMenuHandle 为您的应用程序的帮助菜单的菜单记录在这个参数中返回一个间接指针的拷贝。

说明

HMGetHelpMenuHandle 函数能够在 mh 参数中为您应用程序的帮助菜单的菜单记录返回一个间接指针的拷贝。有了这个间接指针，您就可以使用 AppendMenu 过程或其它有关的菜单管理器的例程给应用程序的帮助菜单增补项目。帮助菜单自动添加划分您的项目和其它帮助菜单命令项目的分割线。

可以通过创建 'HMNU' 资源和指定 kHMHelpMenuID 常数为它的资源标识符，来确保为您的帮助菜单中您的项目定义帮助汽泡。

菜单管理器函数 MenuSelect 和 MenuKey 返回一个结果，其高位字是菜单标识符，低位字是菜单命令项目。当用户在帮助菜单选择一个增补的项目时，MenuSelect 函数（以及 MenuKey 函数，如果用户选择一个具有等效命令组合键的项目）在高位字返回 kHMHelpMenuID 常数。增补菜单命令项目的菜单命令项目编号在低位字返回。苹果公司保留更改帮助菜单中项目数的权利。如果需要确定帮助菜单中的项目数，应调用 CountMItem 函数。

特殊考虑

请不要使用 GetMenuHandle 函数为帮助菜单的菜单记录获取间接指针。GetMenuHandle 可以为全局帮助菜单的菜单记录返回一个间接指针，而不是为您的应用程序专用的帮助菜单的菜单记录返回间接指针。

结果代码

noErr	0	没有错误
paramErr	−50	错误在参数列表中
memFullErr	−108	堆区空间不够
resNotFound	−192	不能读资源
hmHelpManagerNotInited	−855	帮助菜单没有设置

亦请参阅

有关为您的应用程序的帮助菜单添加项目的来例子，以及如何处理用户在帮助菜单中选择的项目，请参阅 3-68 页中的程序清单 3-14 和 3-81 页中的程序清单 3-26。有关为您的应用程序的菜单创建帮助汽泡的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 一书的“帮助管理器”一章。

添加和删除菜单命令项目

您可以使用 InsertResMenu 和 AppendResMenu 过程将一个指定类型的所有资源的名称添加到一个菜单。您可以使用 AppendMenu 或 InsertMenuItem 过程，将您定义的菜单命令项目添加到菜单中。您也可以使用 DeleteMenuItem 过程删除菜单命令项目。在大多数情况下，您不应该在一个已经存在的菜单中插入或删除单个菜单命令项目，除非用户希望更改菜单（例如一个当前打开文稿的程序清单）。

若您使用 AppendMenu 或 InsertMenuItem 过程添加菜单命令项目，您应该在资源中定义您添加的菜单命令项目的文字和其它特征。这会使您的应用程序容易在其它地区本地化。

AppendMenu

使用 AppendMenu 过程为原来使用 NewMenu、GetMenu 或 GetNewMBar 创建的一个菜单增补一个或多个项目。

```
PROCEDURE AppendMenu (menu: MenuHandle; data: Str255);
```

`menu` 一个菜单的菜单记录间接指针，您希望为该菜单增补一个或多个菜单命令项目。

`data` 用来定义一个新的或多个菜单命令项目特征的字符串。注意，在大多数情况下，您应该在一个资源中储存菜单命令项目的文字，这样您的菜单命令项目更容易本地化。AppendMenu 过程按照菜单命令项目在 data 参数中列出的次序增补它们。

说明

AppendMenu过程能够为指定的菜单增补任何定义的菜单命令项目。菜单命令项目添加在菜单的结尾。您可以在data参数中指定任何菜单命令项目的文字和它们的特征。您可以通过在字符串中嵌入元字符来定义菜单命令项目的各种特征。

下面是您可以在data 参数中指定的元字符:

元字符	说明
; 或回车	分隔菜单命令项目
^	当后面是一个图标号码时，定义项目图标。若等效命令组合键字段包含 $1C，该号码解释为一个语系代码。
!	当后面是字符时，定义项目标记。若等效命令组合键包含 $1B，该值则可以解释为这个菜单命令项目子菜单的菜单标识符。
<	当后面是一个或多个字符 B、I、U、O 和 S 时，分别定义项目的字符字形为黑体、斜体、下划线、外形或阴影。
/	当后面是字符时，定义项目的等效命令组合键。后面是 $1B 时，指定该菜单命令项目有一个子菜单。如果希望为菜单命令项目指定一个语系代码、小图标、或缩小图标，可以使用 SetItemCmd 过程将等效组合键域分别设定为 $ 1C、 $ 1D 或 $ 1E。
(	定义菜单命令项目是无效的。

您可以在字符串中指定这些元字符中的任何一个、所有的元字符，或不指定。您指定的元字符在菜单命令项目中不显示。(若要在菜单命令项目的文字中使用这些元字符，首先使用 AppendMenu，至少指定一个字符作为项目文字，然后使用 SetMenuItemText 过程将项目的文字设定到要求的字符串。)

附 注:

如果您使用 AppendMenu 过程添加菜单命令项目，您应该在资源中定义文字和所有的标记或等效命令组合键，因为这样将有助于菜单命令项目的本地化。

您可以将定义菜单命令项目文字的第一个字符指定为连字符，以便创建一条分组线。data参数中的字符串可以是空白(包含一个或多个空格)，但是它不应该是一个空字符串。

如果您不定义菜单命令项目的一个专门的特征，AppendMenu 过程就会为菜单命令项目指派系统设定特征。如果除了菜单命令项目的文字外，您不定义其它任何特征，AppendMenu 过程便将菜单命令项目作为启用项目插入菜单中，它没有图标或标记，使用的是正常字形，也没有等效命令组合键。

无论菜单是否在当前菜单列表中，您都可以使用 AppendMenu 将项目增补到该菜单中。

亦请参阅

有关将项目增补到菜单中的实例，请参阅 3-64 页中的“为菜单添加命令项目”。

InsertMenuItem

使用 InsertMenuItem 过程将一个或多个菜单命令项目插入原来用 NewMenu、GetMenu 或 GetNewMBar 创建的菜单中。

InsertMenuItem 过程和 InsMenuItem 过程是一样的。

```
PROCEDURE InsertMenuItem (theMenu: MenuHandle; itemString: Str255;
                          afterItem: Integer);
```

theMenu　菜单的菜单记录的间接指针，您希望向该菜单添加一个或多个菜单命令项目。

itemString　定义新菜单命令项目特征的一个字符串。注意，在大多数情况下，您应将菜单命令项目的文字储存在一个资源中，这样您的菜单命令项目就更容易本地化。您可以用元字符指定 itemString 参数的内容；InsertMenuItem 过程和 AppendMenu 过程接受同样的元字符。但是，如果您指定多个项目，InsertMenuItem 过程将按照这些项目在 itemString 参数中的相反的顺序插入项目。

afterItem　菜单命令项目的项目编号，新的菜单命令项目将添加在该菜单命令项目之后。在 afterItem 参数中指定 0，新项目插入在第一个菜单命令项目之前，指定一个当前菜单命令项目的项目编号，新菜单命令项目将插入在该菜单命令项目之后；指定一个大于或等于菜单中最后一个项目的号码，新项目将增补在菜单的结尾。

说明

InsertMenuItem 过程可以将所有定义菜单命令项目插入指定菜单。菜单命令项目根据 afterItem 参数指定的位置插入。您可以在 itemString 参数中指定任何菜单命令项目的文字和它们的特征。您可以在字符串中嵌入您指定来定义一个菜单命令项目各种特征的元字符。元字符不在菜单中显示。

下面是您可以在 itemString 参数中指定的元字符:

元字符	说明
；或回车	分隔菜单命令项目
^	当后面是一个图标号码时，定义项目的图标。若等效命令组合键字段包含 ＄1C，该号码解释为一个语系代码。
!	当后面是字符时，定义了项目的标记。若等效命令组合键包含 $1B，该值解释为这个菜单命令项目的子菜单的菜单标识符。
<	当后面是一个或多个字符 B、I、U、O 和 S 时，分别定义项目的字符字形为黑体、斜体、下划线、外形或阴影。
/	当后面是字符时，定义项目的等效命令组合键。后面是 ＄1B 时，指定该菜单命令项目有一个子菜单。若要指定菜单命令项目有一个语系代码、小图标、或缩小图标，应使用 SetItemCmd 过程将等效组合键域分别设定为 ＄1C、＄1D 或 ＄1E。
(	定义菜单命令项目是无效的。

您可以在文字字符串中指定这些元字符中的任何一个，或所有的元字符，或不指定。您指定的元字符在菜单命令项目中不显示。若要在菜单命令项目的文字中使用这些元字符，首先使用 AppendMenu，至少指定一个字符作为项目文字，然后使用 SetMenuItemText 过程将项目的文字设定到要求的字符串。

附注:

若您使用 AppendMenu 过程添加菜单命令项目，您应该在资源中定义文字和所有的标记或等效命令组合键，这样有助于菜单命令项目本地化。

您可以将定义菜单命令项目文字的第一个字符指定为连字符，来创建一条分组线。data 参数中的字符串可以是空白(包含一个或多个空格)，但是它不应该是一个空字符串。

如果您不定义菜单命令项目的一个专门的特征，InsertMenuItem 过程就会为菜单命令项目指派系统设定特征。如果除了菜单命令项目的文字外，您不定义任何特征，InsertMenuItem 过程便将菜单命令项目作为启用项目插入菜单中，它没有图标或标记，使用的是正常字形，也没有等效命令组合键。

无论菜单是否在当前菜单列表中，您都可以使用 InsertMenuItem 将项目增补到该菜单中。

亦请参阅

有关实例，请参阅 3-64 页中的“为菜单添加命令项目”。

DeleteMenuItem

使用 DeleteMenuItem 过程从一个菜单中删除项目。DeleteMenuItem 过程和 DelMenuItem 过程是一样的。

```
PROCEDURE DeleteMenuItem (theMenu: MenuHandle; item: Integer);
```

theMenu　菜单的菜单记录的间接指针，您希望从该菜单删除菜单命令项目。

item　要删除的菜单命令项目的项目编号。如果您指定0或一个比菜单中最后一个项目编号大的数，DeleteMenuItem则不从菜单中删除任何命令项目。

说明

DeleteMenuItem过程可以从菜单中删除一个指定的项目。DeleteMenuItem过程还可以从您的应用程序的菜单色彩信息表中删除项目的菜单命令项目输入(如果有输入的话)。除非用户希望更改菜单（例如列出打开的文稿的菜单），否则您不应该从现有的菜单中删除命令项目。

AppendResMenu

使用AppendResMenu过程在对您的应用程序公开的所有资源文件中搜索给定的资源类型，并将它找到的任何资源的名称增补到指定的菜单。指定的菜单必须是原来用NewMenu、GetMenu或GetNewMBar创建的。

AppendResMenu过程和AddResMenu过程是一样的。

```
PROCEDURE AppendResMenu (theMenu: MenuHandle; theType: ResType);
```

theMenu　菜单的菜单记录间接指针，您希望将AppendResMenu找到的任何给定类型的资源的名称增补到该菜单中。

theType　一个4字符代码，用来标识要搜索的资源类型。

说明

AppendResMenu过程可以在您的应用程序能够打开的所有资源文件中搜索由theType参数指定类型的资源。它将把任何匹配类型的名称增补到指定菜单的结尾。AppendResMenu以找到资源名称的字母顺序进行增补；它并不将已经在菜单中的项目按字母排列。AppendResMenu 过程不把那些名称以句点（.）或百分号（%）开始的资源添加到菜单中。

AppendResMenu过程可以为每个菜单命令项目指派系统设定特征。每个增补的菜单命令项目在菜单中都将作为有效项目出现。这些项目有图标或标记，使用的是正常字形，也没有等效命令组合键。如果希望得到AppendResMenu 增补的项目的名称或更改项目的其它特征，应使用3-130 页的“取得并设定菜单命令项目的外观”中说明的菜单管理器例程。

如果您指定 AppendResMenu 来为您的苹果菜单添加‘DRVR’类型的资源，那么 AppendResMenu 将把苹果菜单命令项目文件夹中每个项目的名称（一个图标）添加到菜单中。

如果您指定 AppendResMenu 来添加‘FONT’或‘FOND’类型的资源，菜单管理器将对它找到的所有字体号码大于 $4000 的资源进行特殊的处理。如果与字体名称有关的语系系统已安装在系统中，那么 AppendResMenu 就会将信息储存在菜单的菜单记录中的 itemDefinitions 阵列（在该项目的 itemIcon 和 itemCmd 域中）中。这将使得菜单管理器能够在正确的语系中显示字体名称。

特殊考虑

AppendResMenu 过程在返回前调用资源管理器过程 SetResLoad（在 load 参数中指定 TRUE）。AppendResMenu 过程将它找到的资源读入存储器。如果您的应用程序在调用其它读资源的例程时，不希望资源管理器将资源数据读入存储器，那么您就需要调用 SetResLoad，并在 AppendResMenu 返回后在 load 参数中指定 FALSE。

亦请参阅

3-69 页中的程序清单 3-15 给出了一个将项目从苹果菜单命令项目文件夹添加到苹果菜单的示例，3-70 页中的程序清单 3-16 给出了将字体名称添加到菜单中的示例。有关资源管理器的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh : More Macintosh Toolbox*)。

InsertResMenu

使用 InsertResMenu 过程在您的应用程序能够打开的所有资源文件中搜索给定的资源类型，并将它找到的任何资源的名称插入指定菜单。指定的菜单必须是原来用 NewMenu、GetMenu, or GetNewMBar。

```
PROCEDURE InsertResMnu (theMenu: MenuHandle; theType: ResType;
                        afterItem: Integer);
```

theMenu　菜单的菜单记录间接指针，您希望将 InsertResMenu 找到的任何给定类型的资源的名称添加到该菜单中。

theType　一个 4 字符代码，用来标识要搜索的资源类型。

afterItem　是用来标识 InsertResMenu 找到的任何给定类型资源的名称插入位置的一个数字。在 afterItem 参数中指定 0，项目插入在第一个菜单命令项目之前，指定已在菜单中的一个菜单命令项目的项目编号，项目将插入指定项目编号之后; 如果您指定一个大于或等于菜单中最后一个项目的号码，则项目插入菜单的结尾。

说明

InsertResMenu过程在对您的应用程序能够打开的所有资源文件中搜索由theType参数指定类型的资源。它将任何找到的给定类型的名称插入指定菜单的指定位置。InsertResMenu 以所找到资源名称的字母顺序进行添加；它并不将菜单中的现有项目按字母顺序排列。

InsertResMenu 过程不把名称以句点（.）或百分号（%）开始的资源添加到菜单中。

InsertResMenu 过程可以为每个菜单命令项目指派系统设定特征。每个增补的菜单命令项目在菜单中都将作为有效项目出现。没有图标或标记，使用的是正常字形，也没有等效命令组合键。如果希望得到 InsertResMenu 增补的项目的名称或更改项目的其它特征，应使用下一节“获取并设定菜单命令项目的外观”中说明的菜单管理器例程。

如果您指定 InsertResMenu 来为您的苹果菜单添加‘DRVR’类型的资源，那么 InsertResMenu 就会将苹果菜单命令项目文件夹中每个项目的名称（一个图标）添加到菜单中。

如果您指定 InsertResMenu 来添加‘FONT’或‘FOND’类型的资源，菜单管理器将对它找到的所有字体号码大于 $4000 的资源进行特殊的处理。如果与字体号码有关的语系系统在当前是激活的，InsertResMenu 将把信息储存在菜单的菜单记录中的 itemDefinitions 阵列（在该项目的 itemIcon 和 itemCmd 域中）中，这使得菜单管理器能够在正确的语系中显示字体名称。

特殊考虑

InsertResMenu 过程在返回前调用资源管理器过程 SetResLoad（在 load 参数中指定 TRUE）。InsertResMenu 过程将它找到的资源读入存储器。如果您的应用程序在调用其它读资源的例程时，不希望资源管理器将资源数据读入存储器，您就需要调用 SetResLoad 并在 InsertResMenu 返回后在 load 参数指定 FALSE。

获取和设定菜单命令项目的外观

您可以使用菜单管理器例程得到一个菜单命令项目的信息。例如，您可以得到一个项目的文字、字形、标记、等效命令组合键、语系代码和所属的图标。您还可以确定一个菜单命令项目是否包含一个子菜单，以及子菜单的菜单标识符。

您可以使用菜单管理器的例程设定一个菜单命令项目的特征，包括将一个子菜单挂接在一个菜单命令项目上。但是，只要可能，您应该在 'MENU' 资源中定义您的应用程序的菜单命令项目。这将有助于您的应用程序在其它地区本地化。

您也可以使用菜单管理器的例程启用或停用菜单命令项目或整个菜单。

EnableItem

使用 EnableItem 过程启用一个菜单命令项目或一个菜单。

```
PROCEDURE EnableItem (theMenu: MenuHandle; item: Integer);
```

`theMenu` 菜单的菜单记录间接指针，该菜单包含要启用的菜单命令项目。

`item` 启用菜单命令项目的项目编号，或者是 0，则启用整个菜单。您不能单独启用一个项目编号大于 31 的菜单命令项目。如果您在 item 参数中指定 0，EnableItem 过程将启用菜单标题和菜单中所有的原来未被单独停用的项目。

说明

EnableItem 过程可以启用一个指定菜单命令项目，使它的外观不再是暗淡的，并使用户能够选择该菜单命令项目。

注意，如果您启用一个菜单，EnableItem 过程将启用菜单标题，但只启用哪些当前不是您的应用程序以前调用 DisableItem 来停用的菜单命令项目，之所以是非停用项目，指定每个项目的项目编号。例如，如果您的应用程序的编辑菜单中所有的项目都被启用了，您可以使用 DisableItem 单个地停用“剪”和“复制”命令。如果您选择将 0 作为 item 参数传送给 DisableItem 来停用整个菜单，菜单和它的所有项目都被停用。然后若您通过将 0 作为 item 参数传送给 EnableItem 来启用整个菜单，该菜单和它的所有项目都被启用，除了“剪”和“复制”命令仍然在停用状态。这时，如果希望启用“剪”和“复制”命令，您必须使用 EnableItem 逐个地启用这两个项目。

如果您的应用程序使用 EnableItem 启用一个菜单，它应该调用 DrawMenuBar 来更新菜单栏的外观。

亦请参阅

有关在菜单中启用项目实例，请参阅 3-58 页的“启用和停用菜单命令项目”。

DisableItem

使用 DisableItem 过程停用一个菜单命令项目或整个菜单。

```
PROCEDURE DisableItem (theMenu: MenuHandle; item: Integer);
```

theMenu　　菜单的菜单记录间接指针，该菜单包含要停用的菜单命令项目。

item　　停用菜单命令项目的项目编号，或者为 0，则停用整个菜单。您不能单独停用一个项目编号大于31的菜单命令项目。如果您在item参数中指定0，DisableItem 过程将停用菜单标题和菜单中所有的项目，包括项目编号大于 31 的菜单命令项目。

说明

DisableItem过程可以停用一个指定的菜单命令项目，使它的外观变得暗淡，并使用户不能够选择该菜单命令项目。

如果您的应用程序使用DisableItem停用一个菜单，它应该调用DrawMenuBar来更新菜单栏的外观。

亦请参阅

有关在菜单停用项目实例，请参阅 3-58 页的“启用和停用菜单命令项目”。

GetMenuItemText

使用GetMenuItemText 过程来获取一个指定菜单命令项目的文字。GetMenuItem-Text 过程和 GetItem 过程是一样的。

```
PROCEDURE GetMenuItemText (theMenu: MenuHandle; item: Integer;
                           VAR itemString: Str255);
```

theMenu　　菜单的菜单记录间接指针，该菜单包括您要为其获取文字的菜单命令项目。

item　　菜单命令项目的项目编号。GetMenuItemText 过程返回该项目的文字。

itemString　　GetMenuItemText 过程在这个参数中返回菜单命令项目的文字。

说明

GetMenuItemText 过程在 itemString 参数中返回指定菜单命令项目的文字。应使用内存管理器的例程来获取有关菜单命令项目其它特征的信息。

SetMenuItemText

使用SetMenuItemText过程将一个指定菜单命令项目的文字设定到一个给定的字符串。SetMenuItemText过程和SetItem过程是一样的。

```
PROCEDURE SetMenuItemText (theMenu: MenuHandle; item: Integer;
                           itemString: Str255);
```

theMenu　菜单的菜单记录间接指针，该菜单包括您要设定文字的菜单命令项目。

item　菜单命令项目的项目编号。SetMenuItemText 过程设定该项目的文字。

itemString　SetMenuItemText 过程根据在itemString参数中指定的字符串设定菜单命令项目的文字。SetMenuItemText过程不能认别元字符，或不设定菜单命令项目的其它特征。itemString参数可以是空白的，但是它不应该是一个空字符串。

说明

SetMenuItemText过程可以将菜单命令项目的文字设定为在itemString参数中指定的文字。SetMenuItemText过程不能认别AppendMenu和InsertMenuItem 过程使用的任何元字符。应使用存储器管理器的例程来设定一个菜单命令项目的其它特征。

如果您使用SetMenuItemText过程来设定菜单命令项目的文字，您应该在一个字符串资源中储存这些文字，这样将便于您的应用程序本地化。

亦请参阅

有关设定一个菜单命令项目文字实例，请参阅 3-59 页中的程序清单 3-9。

GetItemStyle

使用GetItemStyle过程来获取一个指定菜单命令项目中文字的字形。

```
PROCEDURE GetItemStyle (theMenu: MenuHandle; item: Integer;
                        VAR chStyle: Style);
```

theMenu　菜单的菜单记录间接指针，该菜单包括您要为其获取文字体式的菜单命令项目。

item　菜单命令项目的项目编号。GetItemStyle过程返回该项目的文字体式。

csStyle　GetItemStyle 过程在 csStyle 参数中返回该项目的文字体式。csStyle 参数是 Style 数据类型定义的一个集合。

```
TYPE
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

说明

GetItemStyle 过程在 csStyle 参数中返回指定菜单命令项目的文字体式。返回的字形可以是一个或多个由 Style 数据类型定义的字形，如果字形是简明文字，返回的便是空集合。

SetItemStyle

使用 SetItemStyle 过程设定指定菜单命令项目中文字的字形。

```
PROCEDURE SetItemStyle (theMenu: MenuHandle; item: Integer;
                        chStyle: Style);
```

theMenu　菜单的菜单记录间接指针，该菜单包括您要设定文字体式的菜单命令项目。

item　菜单命令项目的项目编号。SetItemStyle 过程设定该项目的文字体式。

csStyle　SetItemStyle 过程根据 csStyle 参数中说明的字形设定该项目的文字体式。csStyle 参数是 Style 数据类型定义的一个集合。

```
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

您可以将字形设定为一个或多个由 Style 数据类型定义的字形，或者您可以通过在 csStyle 参数中指定一个空集合，将它设定为简明文字。

说明

SetItemStyle 过程可以将指定菜单命令项目的文字体式设定为 csStyle 参数定义的一个或多个字形。

亦请参阅

有关设定一个菜单命令项目字形实例，请参阅 3-60 页中的程序清单 3-10.

GetItemMark

使用GetItemMark过程可以获取特定菜单命令项目的标记或与该菜单命令项目相关联的子菜单的菜单标识符。

```
PROCEDURE GetItemMark (theMenu: MenuHandle; item: Integer;
                       VAR markChar: Char);
```

theMenu	是某一菜单的菜单记录的间接指针,该菜单中包含您希望获取其标记或子菜单的菜单项目。
item	菜单命令项目的项目编号。GetItemMark 过程返回此项目的标记，或者，如果此项 目带有与之相连的子菜单，在 markChar 参数中返回子菜单的菜单标识符。
markChar	GetItemMark过程在markChar参数中返回此项目的标记或子菜单。一个菜单项目可以有一个标记或与之挂接的一个子菜单,但不能二者皆有。如果该菜单项目有一个标记字符,GetItemMark过程返回该标记。如果该菜单命令项目有一个与 之相连的子菜单,GetItemMark过程返回该子菜单的菜单标识符。如果该项目没有标记或子菜单,GetItemMark在此参数中返回0。

说明

如果项目有一个标记或子菜单，GetItemMark 过程在 markChar 参中返回该标记或指定菜单命令项目的子菜单的菜单标识符（如果该项目没有标记或子菜单，则返回0)。

SetItemMark

使用SetItemMark 过程可以设定特定菜单命令项目的标记或者更改、设定与一个菜单命令项目相连的子菜单。

```
PROCEDURE SetItemMark (theMenu: MenuHandle; item: Integer;
                       markChar: Char);
```

theMenu	是某一菜单的菜单记录的间接指针,该菜单中包含您希望设定其标记或子菜单的菜单项目。
item	菜单命令项目的项目编号。SetItemMark 过程设定此项目的标记或子菜单。
markChar	SetItemMark过程根据markChar参数中的信息设定该项目的标记或子菜单。

若要设定一个菜单命令项目的标记，可在 markChar 参数中指定标记字符。您也可以使用这些常数中的一个指定该项目没有标记、有一个检查标记（✓）或有一个菱形（◆）作为标记字符:

```
CONST
noMark      = 0;      {no marking character}
checkMark   = $12;    {checkmark}
diamondMark = $13;    {diamond symbol}
```

若要设定与该菜单命令项目相关联的子菜单，可在 markChar 参数中指定该子菜单的菜单标识符。

说明

SetItemMark 过程可以设定标记或指定菜单命令项目的子菜单。

亦请参阅

第 3-61 页上的程序清单 3-11 中关于设定菜单命令项目的标记的实例。

CheckItem

使用CheckItem过程可以将特定菜单命令项目的标记设定为一个检查标记或从一个菜单命令项目中删除一个标记。

```
PROCEDURE CheckItem (theMenu: MenuHandle; item: Integer;
                     checked: Boolean);
```

`theMenu` 是某一菜单的菜单记录的间接指针，该菜单中包含您希望将其标记设定为检查标记或您希望将其标记删除的菜单命令项目。

`item` 菜单命令项目的项目的编号。

`checked` CheckItem 过程根据 checked 参数中的信息可设定或删除该项目的标记。若要将菜单命令项目的标记设定为检查标记，可在checked参数中指定 TRUE。若要从一个菜单项目中删除一个检查标记或任何其它标记，可在 checked 参数中指定 FALSE。

说明

CheckItem过程可以将指定菜单命令项目的标记设定为一个检查标记或从该菜单命令项目中删除任何标记。

亦请参阅

第 3-61 页上程序清单 3 － 11 中关于设定菜单命令项目标记的实例。

GetItemIcon

使用GetItemIcon过程可以获取特定菜单命令项目的图标或语系代码。如果菜单命令项目的等效命令组合键域中包含 $ 1C，则返回的编号代表该菜单命令项目的语系代码。否则，返回的编号代表该项目的图标编号。

```
PROCEDURE GetItemIcon (theMenu: MenuHandle; item: Integer;
                       VAR iconIndex: Byte);
```

theMenu　是某一菜单的菜单记录的间接指针，该菜单包含您希望获取其图标或语系代码的菜单命令项目。

item　菜单命令项目的项目编号。GetItemIcon过程返回此项目的图标编号或语系 代码。

iconIndex　对于在等效命令组合键域中未指定 $ 1C的菜单命令项目，GetItemIcon过程在此参数中返回该项目图标的图标编号。如果该菜单命令项目有一个与之相关联的图标，则在此参数中返回的图标编号为数值1至255，否则为0。将该图标编号加上256，您就可以生成'cicn'、'ICON'或'SICN'资源的资源标识符，该资源用来说明菜单项目的图标。例如，如果GetItemIcon过程在此参数中返回5，那么就由一个图标资源说明该菜单命令项目的图标的资源标识符是261。对于在等效命令组合键域中包含 $ 1C的菜单命令项目，GetItemIcon过程返回菜单命令项目的语系代码。如果安装了对应的语系系统，菜单管理器就使用此语系代码显示该菜单命令项目。

说明

GetItemIcon过程可以在iconIdex参数中返回图标编号或指定菜单命令项目的语系代码。(如果该项目没有图标或语系代码，则返回0)。

SetItemIcon

使用SetItemIcon过程可以设定图标编号或特定菜单命令项目的语系代码。通常您使用当前系统语系显示菜单命令项目；但是，如果需要，您也可以使用SetItemIcon过程来设定一个菜单命令项目的语系代码。若要设定一个项目的语系代码，该项目的等效命令组合键域中包含 $ 1C。如果该等效命令组合键域包含任何其它值，则SetItemIcon过程将此指定的编号解释为该项目的图标编号。

```
PROCEDURE SetItemIcon (theMenu: MenuHandle; item: Integer;
                       iconIndex: Byte);
```

theMenu　是某一菜单的菜单记录的间接指针，该菜单中包含您希望设定其图标（或语系代码）的菜单命令项目。

item　菜单命令项目的项目编号。SetItemIcon 过程可以设定此项目的图标（或语系代码）。

iconIndex　如果菜单命令项目的等效命令组合键域中未包含 $ 1C，则SetItemIcon 过程将该项目图标的图标编号设定为此参数中所定义的编号。您指定的图标编号应为数值1至255（或者，如果项目有一个小图标或缩小图标，则应指定为1至254），如果该项目没有图标，则应指定为0。菜单管理器将图标编号加上256，可以生成'cicn'或'ICON'资源的资源标识符，该资源用来描述菜单命令项目的图标。例如，当菜单管理器需要绘制该项目时，如果您指定5作为iconIndex参数的值，它将寻找资源标识符为261的图标资源。如果该菜单命令项目的等效命令组合键域中包含 $ 1C，则 SetItemIcon 过程将该菜单项目的语系代码设定成 iconIndex参数中定义的编号。如果安装了对应的语系统，则菜单管理器使用指定的语系代码显示该菜单命令项目。您可以在iconIndex参数中指定 0，表示该项目使用当前系统语系并且没有图标编号。

说明

SetItemIcon过程将iconIndex参数中的值设定成图标编号或指定菜单命令项目的语系代码。

亦请参阅

3 - 62 页的“更改菜单命令项目的图标或语系代码”中关于设定一个菜单命令项目的图标的实例。

GetItemCmd

使用 GetItemCmd 过程可以获取一个菜单命令项目的等效命令组合键域值。

```
PROCEDURE GetItemCmd (theMenu: MenuHandle; item: Integer;
                      VAR cmdChar: Char);
```

theMenu　是某一菜单的菜单记录的间接指针，该菜单中包含您希望获取其等效命令组合键域的菜单命令项目。

item　菜单命令项目的项目编号。GetItemCmd 过程返回此项目的等效组合域。

cmdChar　　项目等效命令组合键域的值。菜单管理器使用此数值将等效命令组合键映射到菜单命令或表示该菜单命令项目的特殊特征。如果cmdChar参数中包含＄1B，则菜单命令项目有一个子菜单；数值＄1C表示该项目有一个语系代码；数值＄1D表示菜单管理器缩小该项目的'ICON'资源；数值＄1E表示该项目有一个'SICN'资源。

说明

GetItemCmd过程可以在cmdchar参数中返回指定菜单命令项目的等效命令组合键域中的值，如果该项目没有等效命令组合键、子菜单、语系代码、缩小图标或小图标，则返回0)。

SetItemCmd

使用SetItemCmd过程可以设定一个菜单命令项目的等效命令组合键域的值。通常您可在'MENU'资源中定义等效命令组合键和菜单命令项目的其它特征，而不是使用SetItemCmd过程来定义。

```
PROCEDURE SetItemCmd (theMenu: MenuHandle; item: Integer;
                      cmdChar: Char);
```

theMenu　　是某一菜单的菜单记录的间接指针，该菜单中包含您希望设定其等效命令组合键域的菜单命令项目。

item　　菜单命令项目的项目编号。SetItemCmd过程将此项目的等效命令组合键域设定为cmdChar参数中指定的值。

cmdChar　　该项目的等效命令组合键域的值。菜单管理器使用此数值将等效命令组合键映射到菜单命令或定义该菜单命令项目的特殊特征。若要表示该菜单命令项目有一个子菜单，就在cmdChar参数中指定＄1B；若要表示该项目有一个语系代码就指定数值＄1C；若要表示菜单管理器应将项目的'ICON'资源缩小成小图标大小，就指定数值＄1D；若要表示该项目有一个'SICN'资源，就指定数值＄1E。数值＄01至＄1A，以及＄1F和＄20供苹果公司使用。您不可在cmdChar参数中使用这些保留值。

说明

SetItemCmd过程可以在cmdChar参数中设定指定菜单命令项目的等效命令组合键域中的值（如果该项目没有等效命令组合键、子菜单、语系代码、缩小图标或小图标，您可以指定0）。如果您指定该项目有一个子菜单，就应提供该子菜单的菜单标识符作为该项目的标记字符。如果您指定该项目有一个语系代码。就应在菜单命令项目的图标域中提供语系代码。如果您指定该项目有一个'SICN'或缩小的'ICON'资源，就应在该项目的图标域中提供图标编号。

排列菜单

如果在菜单列表中您不再需要某个菜单，就可以使用DeletMenu删除该菜单。然后，如果您使用NewMenu创建了该菜单，就应该使用DisposeMenu过程释放与该菜单相关联的存储器；否则，就应使用资源管理器的过程ReleaseResource。请参阅《Inside Macintosh工具箱扩展》(*Inside Macintosh: More Macintosh toolbox*)的"资源管理器"一章中关于ReleaseResource例程的说明。

DisposeMenu

若要释放与一个菜单关联的数据结构所占用的存储器，请使用DisposeMenu过程或资源管理器的过程ReleaseResource。如果您使用NewMenu创建了该菜单，就可使用DisposeMenu；如果您使用GetMenu创建了该菜单或使用GetNewBar读资源，就可使用ReleaseResource。

在调用DisposeMenu过程之前，您应用使用DeleteMenu或clearMenuBar从当前菜单列表中删除该菜单。

```
PROCEDURE DisposeMenu (theMenu: MenuHandle);
```

`theMenu` 是某一菜单的菜单记录的间接指针，该菜单是您希望进行排列的菜单。

说明

DisposeMenu过程可以释放指定菜单的菜单记录所占用的存储器。当DisposeMenu返回后，您在参数theMenu中传送的间接指针将失效。

亦请参阅

若要从当前菜单列表中删除一个菜单，请参阅第3－109页中关于DeleteMenu过程的说明。

计算菜单中的项目数

如果您的应用程序需要计算一个菜单中项目的数目—例如Font菜单或Help菜单等菜单命令项目的数目可变的菜单—就可使用CountMItems函数。

CountMItems

您可以使用 CountMItems 函数计算一个菜单中项目的数目。

```
FUNCTION CountMItems (theMenu: MenuHandle): Integer;
```

theMenu 是某一菜单的菜单记录的间接指针，您的应用程序需要计算该菜单的项目。

说明

CountMItems 函数可以计算指定菜单中的项目的数目，当此函数计算出该菜单中有多少项目时将返回。

醒目显示菜单栏

您可以使用 FlashMenuBar 过程来醒目显示（反色）一个菜单标题或整个菜单栏。(HiliteMenu 过程仅醒目显示菜单标题。) 多数情况下，您的应用程序不应醒目显示菜单栏；使用 HiliteMenu 可醒目显示一个菜单标题。

用户可使用通用控制板来设定一个应用菜单命令项目闪烁的次数。当用户选择一个应用菜单命令项目时，SetMenuFlash 过程可用来控制菜单命令项目闪烁的次数；通常您不应更改用户选择的设定状态。

FlashMenuBar

使用 FlashMenuBar 过程可以醒目显示（反色）一个菜单标题或整个菜单栏。若要使其菜单栏闪烁，您可以对一横行两次调用 FlashMenuBar。

```
PROCEDURE FlashMenuBar (menuID: Integer);
```

MenuID 您希望反色显示其标题的菜单的菜单标识符。在此参数中使用0可以反色显示整个菜单栏。如果指定的菜单标识符不存在于当前菜单列表中，FlashMenuBar 过程就反色显示整个菜单栏。

说明

FlashMenuBar过程可以反色显示指定菜单的标题或反色显示菜单栏。若要防止不希望的彩色在菜单栏中出现，当反色显示整个菜单栏时不应调用FlashMenuBar来反色显示菜单标题。

每次仅可以反色显示一个菜单标题。如果当前没有醒目显示的菜单，则利用特定菜单标识符调用 FlashMenuBar 就可以反色显示该菜单的标题。如果再次调用FlashMenuBar以指定另一个菜单标识符，它不同于先前反色显示菜单标题的菜单标识符，则 FlashMenuBar 将先前醒目显示的菜单还原到正常显示，然后反色显示指定菜单的标题。

亦请参阅

根据第 3 － 119 页的介绍您也可以使用 HiliteMenu 过程醒目显示一个菜单。

SetMenuFlash

当用户选择一个启用菜单命令项目时使用SetMenuFlash过程可以设定菜单命令项目闪烁的次数。用户可使用通用控制板设定此值，多数情况下，您的应用程序不应更改用户设定的值。

```
PROCEDURE SetMenuFlash (count: Integer);
```

`Count` 用户选择一个启用菜单命令项目时它应闪烁的次数。通用控制板初始设定此值为 3。若计数为 0 则停用闪烁。此值如果大于 3，用户的操作就会减慢并造成混乱。

说明

当用户选择一个启用项目时，SetMenuFlash过程可设定菜单管理器使一个菜单命令项目闪烁的次数。

菜单命令项目闪烁的外观由该菜单的菜单定义过程决定。

汇编语言信息

全局变量MenuFlash中包含用户所选择的菜单命令项目的当前闪烁计数（次数）。

重新计算菜单尺寸

每当一个菜单的内容有所变更时，菜单管理器使用CalcMenuSize过程重算该菜单的尺寸。多数情况下，您的应用程序不需要使用 CalcMenuSize 过程。

CalcMenuSize

CalcMenuSiae过程可以重算一个菜单的水平和垂直尺寸并将此新值存储在菜单记录的 menuWidth 和 menuHeight 域中。

```
PROCEDURE CalcMenuSize (theMenu: MenuHandle);
```

theMenu　　是某一菜单的菜单记录的间接指针，该菜单需要重新计算尺寸。

说明

CalcMenuSize 过程使用指定菜单的菜单定义过程来计算该菜单的尺寸。

管理菜单色彩信息表中的输入

菜单管理器可以将有关某一应用程序的菜单的彩色信息保留在菜单色彩信息表中。标准菜单定义过程可以定义菜单栏、菜单标题、菜单命令项目的文字和特征的标准彩色，以及菜单显示的背景色。通过在应用程序的菜单色彩信息表中添加输入，您就可以更改这些彩色。但是请注意，多数情况下，您的应用程序应使用系统设定色彩显示其菜单。

如果您希望使用非系统设定色彩来显示应用程序的菜单栏和菜单，您可以将资源标识符为 0 的一个 'mctb' 资源作为应用程序的一个资源。（或者您可以用其资源标识符与 'MENU' 资源标识符相同的一个 'mctb' 资源定义单个菜单的彩色输入。）您亦可以使用 SetMCEntries和DeleteMCEntries过程对应用程序的菜单色彩信息表添加或删除输入。您可以使用 GetMCEntry 函数获取某一输入的信息。若要获取或设定应用程序的菜单色彩信息表，您可使用 GetMCInfo 函数或 SetMCInfo 过程。若要排列应用程序的菜单色彩信息表，您可使用 DisposeMCInfo 过程。

请注意，菜单色彩信息表所用的格式不同于标准的彩色表格式。从第 3 － 98 页开始的“菜单色彩信息表记录”中详细介绍了菜单色彩信息表的格式。

GetMCInfo

您可以使用 GetMCInfo 函数获取应用程序的某一菜单色彩信息表拷贝的间接指针。

```
FUNCTION GetMCInfo: MCTableHandle;
```

说明

GetMCInfo函数可以创建应用程序的一份菜单色彩信息表并对该份信息表返回一个间接指针。如果拷贝失败，则 GetMCInfo 返回 NIL。

亦请参阅

3 – 98 页的“菜单色彩信息表记录”中关于菜单色彩信息表格式的说明。

SetMCInfo

您可使用 SetMCInfo 过程设定应用程序的菜单色彩信息表。

```
PROCEDURE SetMCInfo (menuCTbl: MCTableHandle);
```

menuCTb1　　是某一菜单色彩信息表的间接指针。

说明

SetMCInfo过程可将menuCTb1参数指定的表拷贝到应用程序的菜单色彩信息表中。如果拷贝成功，则 SetMCInfo 过程负责排列应用程序的当前菜单色彩信息表，因此您的应用程序不需要直接地排列该当前表。

若要确定SetMCInfo过程是否成功地拷贝了该表则您的应用程序应调用存储器管理器的函数 MemError，如果 SetMCInfo 过程拷贝该表失败，则不能排列当前菜单色彩信息表，并且 MemError 函数返回一个非零的结果代码。如果 SetMCInfo 过程能够成功地拷贝该表，则当前菜单彩彩色信息表可被排列，并且 MemError 函数返回 noErr 结果代码。如果菜单色彩信息表指定新的菜单栏彩色或新的菜单的菜单标题彩色，您的应用程序应在调用 SetMCInfo 之后调用 DrawMenuBar。

请注意，GetNewMBar 不存储应用程序的当前菜单色彩信息表。如果您的应用程序更改了菜单栏，只需先调用 GetNewMBar，然后调用 GetMCInfo，之后再调用 SetMCInfo，您就可以存储并还原应用程序的当前菜单色彩信息表。

亦请参阅

3 – 98 页的“菜单色彩信息表记录”中关于菜单色彩信息表格式的说明。有关使用 GetMCInfo 和 SetMCInfo 例程以存储并还原菜单彩色信息的实例，请参阅第 3 – 52 页上的程序清单 3 – 6。有关 MemError 函数的说明请参阅《Inside Macintosh 内存》(*Inside Macintosh: Memory*)。

DisposeMCInfo

您可以使用 DisposeMCInfo 过程排列菜单色彩信息表。DisposeMCInfo 过程亦称 DispMCInfo 过程。

```
PROCEDURE DisposeMCInfo (menuCTbl: MCTableHandle);
```

menuCTb1　　是菜单色彩信息表的间接指针。

说明

DisposeMCInfo 过程可以排列 menuCTb1 参数引用的菜单色彩信息表。

GetMCEntry

您可以使用 GetMCEntry 函数返回应用程序的菜单色彩信息表中输入的信息。您可获取菜单栏输入、菜单标题输入、或菜单命令项目输入的信息。

```
FUNCTION GetMCEntry (menuID: Integer; menuItem: Integer)
                     : MCEntryPtr;
```

menuID　　GetMCEntry 函数应利用此菜单标识符返回菜单色彩信息表的信息。在 menuID 参数（和 menuItem 参数）中指定 0，就可以获取菜单栏输入。在 menuID 参数中指定当前菜单列表中一个菜单的菜单标识符并在 menuItem 参数中指定 0，就可以获取一个特定的菜单标题输入。在 menuID 参数中指定当前菜单列表中一个菜单的菜单标识符并在 menuItem 参数中指定一个项目编号，就可以获取一个特定的菜单项目输入。

menuITEm　　GetMCEntry 函数应利用此菜单命令项目来返回菜单色彩信息表的信息。如果您在此参数中指定。则 GetMCEntry 根据 menuID 参数的值或返回菜单栏输入，或返回菜单标题输入。如果您在此参数中指定一个菜单命令项目的项目编号，并在 menuID 参数中指定当前菜单列表中一个菜单的菜单标识符，则 GetMCEntry 返回一个特定的菜单命令项目输入。

说明

GetMCEntry 函数根据 menuID 和 menuItem 参数中指定的值可以返回一个菜单栏输入、菜单标题输入或菜单命令项目输入。如果 GetMCEntry 函数在应用程序的菜单色彩信息表中查找到指定的输入，则返回一个指向某一数据类型记录 MCEntry 的指针。如果未查找到指定的输入，则 GetMCEntry 返回 NIL。

▲ **警告**

菜单色彩信息表是可重定位的，因此由GetMCEntry函数返回的指针对于那些可能会移动或清除存储器的例程也许无效。如果需要，您的应用程序应拷贝一份菜单彩色输入记录。▲

亦请参阅

3 - 98页的“菜单色彩信息表记录”中介绍了菜单色彩信息表中的输入。

SetMCEntries

您可以使用SetMCEntries过程在应用程序的菜单色彩信息表中设定输入。您设定应用程序的任一个或全部菜单命令项目输入和菜单标题输入或菜单栏输入。

```
PROCEDURE SetMCEntries (numEntries: Integer;
                        menuCEntries: MCTablePtr);
```

`numEntries` 是在菜单彩色输入记录数组中包含的输入数目。

`menuCEntries` 指向菜单彩色输入记录数组的指针。在numEntries参数中指定该数组中的记录数目。

说明

SetMCEntries过程根据菜单彩色输入记录中指定的值可以设定任何指定的菜单栏输入、菜单标题输入、或菜单命令项目输入。如果指定的菜单彩色输入中已存在一个输入，则SetMCEntries过程在应用程序的菜单色彩信息表中用新的值来更新该输入。如果不存在该输入，则将新值添加到应用程序的菜单色彩信息表中。

如果添加的任何输入指定一个新的菜单栏彩色或新的菜单标题彩色，则应用程序就应调用DrawMenuBar，以便用新的彩色更新该菜单栏。

特殊考虑

SetMCEntries过程可能会移动或清除存储器。您的应用程序在调用SetMCEntries之前应确保由menuCEntries参数指定的数组是不可重定位的。

亦请参阅

3 - 98 页的“菜单色彩信息表记录”中介绍了菜单色彩信息表中的输入。

DeleteMCEntries

您可以使用DeleteMCEntries过程从应用程序的菜单色彩信息表中删除某个或所有的输入。您可删除一个菜单命令项目输入、菜单标题输入、菜单栏输入或指定菜单的全部菜单命令项目输入。DeleteMCEntries 过程亦称 DelMCEntries 过程。

```
PROCEDURE DeleteMCEntries (menuID: Integer; menuItem: Integer);
```

`menuID` DeleteMCEntries过程应利用此菜单标识符来确定从菜单色彩信息表中删除哪个输入项。在 menuID 参数（和 menuItem 参数）中指定 0 可以删除菜单栏输入。在menuID参数中指定当前菜单列表中某个菜单的菜单标识符并在 menuItem 参数中指定 0，可以删除一个特定的菜单标题输入。在menuID参数中指定当前菜单列表中某个菜单的菜单标识符和在menuItem参数中指定一个项目编号，可以删除一个特定的菜单命令项目输入。

`menuItem` DeleteMCEntries过程应利用此菜单命令项目来确定从菜单色彩信息表中删除哪个输入。如果您在此参数中指定 0，DeleteMCEntries 则根据 menuID 参数的值，删除菜单栏输入或删除菜单标题输入。如果您在此参数中指定一个菜单项目的项目编号，并且在menuID参数中指定当前菜单列表中某个菜单的菜单标识符，DeleteMCEntries就删除一个特定的菜单命令项目输入。您也可以使用 mctAllItems（= -98）常数从应用程序的菜单色彩信息表中删除特定菜单的全部菜单命令项目输入。

说明

DeleteMCEntries 过程根据 menuID 和 menuItem 参数中指定的值可以删除某个菜单栏输入、菜单标题输入、菜单命令项目输入或一个给定菜单的全部菜单命令项目输入。如果DeleteMCEntries函数在应用程序的菜单色彩信息表中未查找到指定的输入，则它不删除该输入。您的应用程序不应删除其菜单彩彩色信息表中的最后一个输入。

如果某个被删除的输入更改了该菜单栏彩色或菜单标题彩色，您的应用程序应调用 DrawMenuBar 更新该菜单栏。

应用程序定义的例程

苹果公司提供一个标准的菜单定义过程和标准的菜单栏定义函数。菜单管理器使用菜单定义过程和菜单栏定义函数来显示菜单及菜单栏并执行其中的基本操作。虽然菜单管理器允许提供您自己的菜单栏定义函数，但苹果公司建议您使用标准的菜单栏定义函数。同样，在多数情况下标准的菜单定义过程应能够满足多数应用程序的需要。但是，如果您的应用程序有特殊需要，则可选择提供您自己的菜单定义过程。如果您采取这种作法，在定义菜单定义过程时，应尽量参照菜单的标准做法。如果是定义您自己的菜单，您应遵循本章及《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 中介绍的准则来做。

菜单定义过程

菜单管理器使用一个菜单的菜单定义过程来绘制菜单中的菜单命令项目、确定用户从菜单中选择了哪个项目以及计算菜单的尺寸。如果提供了您自己的菜单定义过程，它也能够执行这些任务。

苹果公司可提供标准菜单定义过程，并作为资源存储在系统文件中。标准菜单定义过程称为'MDEF'资源，其资源标识符是0。当定义您自己的菜单时，您可以指定菜单定义过程，以便菜单管理器能够使用该过程来管理这些菜单。您通常希望在应用程序中使用标准的菜单定义过程。但是，如果标准的菜单定义过程不能提供您需要的功能（例如，如果您希望在菜单中包含更多的图形），那么您可以选择写您自己的菜单定义过程。

MyMenuDef

如果您需要使一个菜单具有特殊功能，而标准菜单定义过程又不提供这些功能，那么您可以提供您自己的菜单定义过程。本节介绍如何定义您自己的菜单定义过程、如何定义由菜单管理器传送到这一过程中的参数以及您的过程应执行的一般动作。

```
PROCEDURE MyMenuDef (message: Integer; theMenu: MenuHandle;
                     VAR menuRect: Rect; hitPt: Point;
                     VAR whichItem: Integer);
```

`message` 标识菜单定义过程应执行的操作的编号。此 message 参数可以包含这些值中的任一个。

```
CONST
      mDrawMsg        = 0; {draw the menu}
      mChooseMsg      = 1; {tell which item was chosen }
                           { and highlight it}
      mSizeMsg        = 2; {calculate menu dimensions}
      mPopUpMsg       = 3; {calculate rectangle of }
                           { the pop-up box}
```

您的菜单定义过程不应对上述四个常数以外的任何值作出响应。

`theMenu` 是某一操作涉及到的菜单记录间接指针。

`menuRect` 是菜单所在的矩形（在全局坐标中）；仅当message参数中的值是mDrawMsg或mChooseMsg常数时，菜单管理器才向菜单定义过程提供此信息。

当message参数中的值是mPopUpMsg常数时，菜单定义过程应在此参数中计算并返回弹出式方框的尺寸。当message参数中的值是mSizeMsg常数时，菜单定义过程应计算菜单矩形的水平和垂直尺寸，并将这些值存储在该菜单记录的menuWidth和menuHeight域中。

`hitpt` 即鼠标位置（在全局坐标中）。当message参数中的值是mChooseMsg或mPopUpMsg常数时，菜单管理器向菜单定义过程提供此参数中的信息。当菜单定义过程接收到message参数中的mChooseMsg常数时，它应确定hitpt参数中指定的鼠标位置是否在一个启用菜单命令项目中，并且适当地醒目显示或不醒目显示whichItem参数中的指定项目。当菜单定义过程接收到massage参数中的mPopUpMsg常数时，hitpt参数中包含关闭的弹出式方框的左上坐标，您的过程可以用它来计算打开弹出式方框的矩形。

`whichItem` 是从该菜单中选择的最后一个项目的项目编号（如果未选择项目，它就是0）。当message参数中的值是mChooseMsg常数时，菜单管理器向菜单定义过程提供此参数中的信息。当菜单定义过程接收到message参数中的mChooseMsg常数时，它应确定在hitpt参数中指定的鼠标位置是否在一个启用菜单命令项目中。如果是，菜单定义过程应不醒目显示whichItem参数指定的项目，而是突出显示新项目，并且在whichItem中返回新的项目编号。如果鼠标位置不在一个启用菜单命令项目中，菜单定义过程应不再醒目显示whichItem参数指定的项目，并　　且在whichItem参数中返回0。

说明

每当菜单管理器需要您的定义过程在特定菜单中执行某个动作时，它就调用您的菜单定义过程。message 参数的值决定了您的菜单定义过程应执行的动作。

如果您提供自己的菜单定义过程，可将它存储在类型'MDEF'的资源中，并且将其资源标识符包含在使用您自己的定义过程的每个菜单说明中。如果您使用GetMenu（或GetNewMBar）创建一个菜单，则菜单管理器将该菜单定义过程读出来放入存储器，并且将其间接指针存储在该菜单的菜单记录的 menuProc 域中。

如果您使用 NewMenu 创建一个菜单，则菜单管理器在该菜单的菜单记录的menuProc域中存储一个标准菜单定义过程间接指针。此时，必须将用于您自己的过程的间接指针取代 menuProc 域中的值，然后调用 CalcMenuSize 过程。如果您的菜单定义过程是在一个资源文件中，您可以使用资源管理器获取其间接指针，以便将它从该资源文件中读出来放入存储器。但是请注意，通常您应将菜单存储在资源中（而不是使用 NewMenu），以使您的应用程序更易于本地化。请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 的“资源管理器”一章中关于资源管理器的说明。

菜单定义过程负责绘制菜单内容及其菜单命令项目，确定光标是否位于被显示的菜单中，醒目显示和不醒目显示菜单命令项目以及计算菜单的尺寸。

当菜单管理器请求您的菜单定义过程执行一个菜单上的动作时，它向您的过程提供一个用于其菜单记录的间接指针。这样，您的过程就可以存取菜单记录中的数据并使用该菜单记录的变量数据分区中的任何数据来恰当地处理菜单命令项目。

当一个应用程序调用 GetMenu 或 GetNewMBar 后，菜单管理器就可创建一个菜单，然后根据其资源定义来填写菜单记录的 menuID、menuProc、enableFlags、menuTitle 和 itemDefinitions 域。如果该菜单是由您的菜单定义过程来管理，菜单管理器就调用您的过程（指定 mSizeMsg）来计算和填写菜单记录的 menuHeight 和 menuWidth 域。该菜单记录当中的可变长度域（itemDefinitions）用来说明菜单命令项目。您的菜单定义过程可以选择任何方式定义和使用这一可变长度数据。

对于不是作为控制部件实现的弹出式菜单，菜单管理器使用菜单定义过程来支持弹出式菜单。如果您的菜单定义过程支持弹出式菜单，它应相应地响应 mPopUpMsg 常数。

菜单管理器在 message 参数中指定 mPopUpMsg 常数，并且每当需要计算由您的菜单定义过程管理的弹出式菜单矩形（由弹出式方框定界）时，它调用您的菜单定义过程。参数 theMenu 中包含用于弹出式菜单的菜单记录的间接指针，hitpt 参数中包含弹出式方框的左上坐标，whichItem 参数中包含先前选择的项目。您的菜单定义过程应计算弹出式菜单出现的矩形，并且在 menuRect 参数中返回此矩形。如果该菜单太大而需要滚动，则在 whichItem 参数中返回实际的菜单顶部。对于弹出式菜单，您的菜单定义过程也必须将弹出式菜单的滚动信息放置于全局变量 TopMenuItem 和 AtMenuBottom 中。在 TopMenuItem 中放置可滚动菜单项部的像素值，在 AtMenuBottom 中放置可滚动菜单底部的像素值。

附　注

您的菜单定义过程不应假定 A5 寄存器已设定好，因此您的过程不能引用任何 QuickDraw 全局变量。◆

亦请参阅

在接收到 message 参数中的 mDrawMsg、mChooseMsg、或 mSizeMsg 常数时，关于您的菜单定义过程应如何响应的详细说明，请参阅从第 3 － 87 页的“编写您自己的菜单定义过程”。

资源

本节介绍菜单（'MENU'）资源、菜单栏（'MBAR'）资源和菜单色彩信息表（'mctb'）资源。通常您应使用 'MENU' 资源定义菜单；用一个 'MBAR' 资源定义菜单栏中的菜单以及使用 GetNewMBar 函数来读菜单和菜单栏的说明。

如果您希望在一个菜单中使用非系统设定的彩色，您可以提供具有相同资源标识符的 'mctb' 资源作为与其对应的 'MENU' 资源，或者提供一个带有资源标识符为 0 的 'mctb' 资源，以便为您所有的菜单和菜单栏定义彩色。

如果选择提供您自己的菜单定义过程，您应将您的例程存储在一个 'MDEF' 资源中。

若要创建一个 'MENU'、'BAAR'、或 'mctb' 资源，您可以在一个输入文件中指定该资源说明并使用资源编译程序（如 Rez）编译该资源，也可以使用工具（如 ResEdit）来直接在资源文件中创建您的资源。本节介绍由 Rez 资源编译程序编译后的结构这些资源。如果您对创建这些资源的 Rez 输入文件感兴趣，请参阅从第 3 － 41 页开始的“使用菜单管理程序”中的详细说明。

菜单资源

您可以在 'MENU' 资源中提供菜单说明，并且使用 GetMenu 函数或 GetNewMBar 函数（如果您同时提供 'MBAR' 资源）来读您的菜单说明。在读资源描述后，菜单管理器可将有关特定菜单的信息存储在菜单记录中。

▲ 警告

在资源中的菜单必须是不可清除的。▲

图 3 － 37 是经编译的'MENU'资源的格式。关于 Rez 输入格式中的'MENU'资源描述，请参阅第 3 － 43 页上的程序清单 3 － 1。

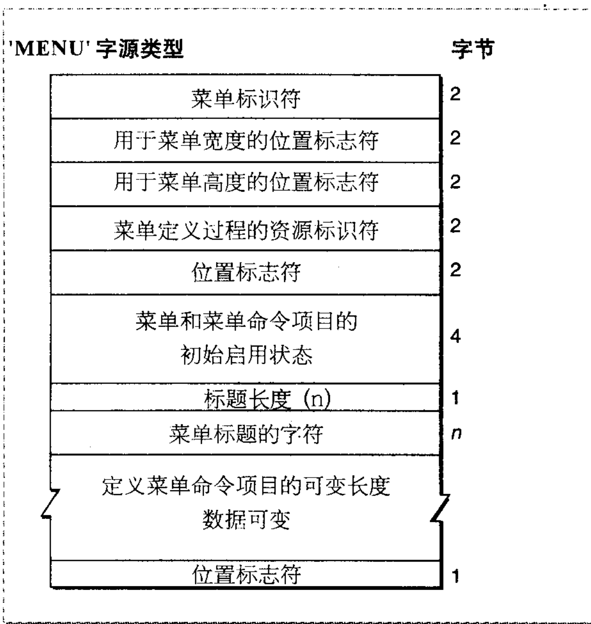


图 3-37　经编译的菜单（'MENU'）资源结构

'MENU'资源的编译版本包含下列元素：

- 菜单标识符。在您的应用程序中每个菜单应有一个不重复的菜单标识符。请注意，菜单标识符不一定要与资源标识符匹配，尽量按照惯例，多数应用程序为一个菜单的资源标识符和菜单标识符指派相同的编号。一个负菜单标识符表示一个菜单属于桌面附件（桌面附件的子菜单除外）。从 1 至 235 的菜单标识符表示一个应用程序的菜单（或子菜单）；从 236 至 255 的菜单标识符表示一个桌面附件的子菜单。苹果公司保留 0 的菜单标识符

- 用于菜单宽度和高度的位置标志符（含有 0 的两个整数）。菜单管理器读资源数据之后，请求菜单的菜单定义过程计算该菜单的宽度和高度，并将这些值存储在该菜单记录的 menuWidth 和 menuHeight 域中

- 菜单的菜单定义过程的资源标识符。如果这里出现整数 0（由 Rez 输入文件中 textMenuProc 常数指定），菜单管理器就使用标准菜单定义过程来管理菜单。如果您提供自己的菜单定义过程，其资源标识符应出现在这些字节中。在读菜单的资源数据后，如果需要，菜单管理器可读该菜单定义过程。菜单管理器在菜单记录的 menuProc 域中为菜单的菜单定义过程存储一个间接指针

- 位置标志符（包含0的一个整数）

- 菜单和前31个菜单命令项目的初始启用状态。这是一个32位的值，其中1 - 31位表示对应的菜单命令项目是启用的或停用的，0位表示菜单是启用的或停用的。当创建一个菜单时，菜单管理器自动地启用位数大于31的菜单命令项目

- 菜单标题的长度（以字节计算）

- 菜单的标题

- 说明菜单命令项目的可变长度数据。如果您提供自己的定义过程，则可根据过程的需要定义和提供此可变长度数据。菜单管理器只是读用于每个菜单命令项目的数据并将它作为变量数据存储在菜单记录的尾部。菜单定义过程负责解释该数据的内容。例如，标准菜单定义过程根据下列段落给出的说明解释此数据

- 位置标志符（1字节，包含0）表示菜单命令项目定义的结尾

如果使用标准菜单定义过程，您的'MENU'资源应以此方式说明菜单命令项目。对于每个菜单命令项目，您需要提供其文字、图标编号、等效命令组合键或其它值（$1B表示菜单命令项目有一个子菜单，$1C表示用于项目文字的系统语系的语系代码，$1D表示项目的图标应缩小，$1E表示应使用'SICN'图标），还包括标记菜单命令项目的字符或菜单命令项目的子菜单的菜单标识符，以及菜单命令项目文字的字形。如果一个项目没有特殊特征，就为该特征指定0。图3 - 38是经编译的'MENU'资源可变长度数据分区，该资源使用标准菜单定义过程。

在'MENU'资源中的可变长度数据（对于每个菜单命令项目）	字节
菜单命令项目文字的长度	1
菜单命令项目的文字	*m*
图标编号、语系代码、或0	1
等效命令组合键、$1B、$1C、$1D、$1E、或0	1
标记字符或子菜单的菜单标识符，或0	1
菜单命令项目的字形	1

图3-38　说明菜单命令项目的可变长度数据，由标准菜单定义过程定义

使用标准菜单定义过程的'MENU'资源的编译版本，其可变长度数据分区包含下列元素：

- 菜单命令项目文字的长度（以字节计算）
- 菜单命令项目的文字
- 图标编号、语系代码或如果菜单命令项目不包含图标并使用系统语系时，为0（由Rez输入文件中noicon常数指定的）。图标编号是从1至255的编号（对于小图标及缩小图标为1至254）。菜单管理器对图标编号加上256就可以生成菜单命令项目图标的资源标识符。如果某个菜单命令项目有一个图标，您还应提供其资源标识符等于图标编号加256的'cicn'或'ICON'资源。如果您希望菜单管理器将'ICON'资源缩小至小图标的大小，那么也要等效命令组合键域提供值＄1D。如果您提供一个'SICN'资源，就要在等效命令组合键域中提供＄1E。否则，菜单管理器将首先寻找带有计算出的资源标识符的'cicn'资源并使用该图标。如果您希望菜单管理器用非系统语系绘制项目的文字，就要在此处指定语系代码，并且还在等效命令组合键域中提供＄1C。如果安装了指定的语系系统，菜单管理器就使用语系绘制项目的文字。用非系统语系绘制的项目不能同时含有图标
- 等效命令组合键（指定为1字节字符），如果项目有一个子菜单，则该值为＄1B（由Rez输入文件中常数hierarchicalMenu指定的），如果项目使用非系统语系，则该值为＄1C，或者，如果项目没有等效命令组合键或子菜单并且使用系统语系，则该值为0（由Rez输入文件中nokey常数指定的）。一个菜单命令项目可以有等效命令组合键、子菜单、小图标、缩小图标或语系代码，但只能有其中一种特征。对于包含图标的项目，如果您希望菜单管理器将'ICON'资源缩小为小图标大小，您可以在此域中提供＄1D。如果您希望菜单管理器将一个'SICN'资源用于项目的图标，则应提供＄1E。值＄01至＄1A以及＄1F和＄20是苹果公司保留使用的；您的应用程序不应使用此域中任何这些保留值
- 标记字符、项目子菜单的菜单标识符，如果项目没有标记或子菜单，则该值为0（由Rez输入文件中nomark常数指定）。一个菜单命令项目可以有一个标记或一个子菜单，但不能二者皆有。应用程序的子菜单的菜单标识符应是1至235；桌面附件的子菜单的菜单标识符应是236至255
- 菜单命令项目的字形。在Rez输入文件中可以使用常数bold、italic、plain、outline和shadow来定义它们对应的字形

如果您提供您自己的菜单定义过程，您应将相同的格式用于您的菜单资源描述，如图3-37所示。您可以使用相同的格式或选择介绍菜单命令项目的格式。您也可以使用您所选择的菜单记录的enableFlags域中1 － 31位；但是，0位必须用于表示菜单是启用的还是停用的。

菜单栏资源

您可以在一个 'MBAR' 资源中说明菜单栏中菜单的顺序和编号，并且您可以在 'MENU' 资源中说明您的菜单。如果您采取这种作法，就可使用 GetNewMBar 函数读菜单的说明并创建新的菜单列表。菜单管理器在菜单列表中可存储应用程序的菜单栏信息。图 3 - 39 是经编译的 'MBAR' 资源的格式。(请参阅第 3 - 49 页上的程序清单 3 - 4 中关于 Rez 输入格式的 'MBAR' 资源描述。)

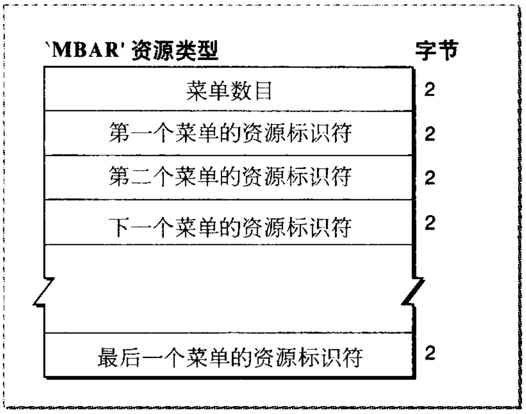


图 3-39　经编译的菜单栏（'MBAR'）资源的结构

一个 'MBAR' 资源的编译版本包含下列元素：

- 此菜单栏说明的菜单数目
- 资源标识符的可变编号（其数目应与前 2 个字节中声明的数目相匹配）；每个资源标识符应标识一个 'MENU' 资源

如果您使用 GetNewMBar 函数，则菜单管理器根据菜单在 'MBAR' 资源中出现的顺序，将菜单放置在菜单栏中。

菜单色彩信息表资源

若要在菜单中使用非系统设定的彩色，就应提供与对应 'MENU' 资源有相同的资源标识符的一个菜单色彩信息表（'mctb'）资源。您也可以选择提供带有资源标识符为 0 的一个 'mctb' 资源来定义所有菜单和菜单栏彩色。请注意，您通常应使用由菜单管理器提供的系统设定色彩。

菜单管理器可将应用程序的菜单栏信息存储在菜单色彩信息表中。如果您提供带有资源标识符为0的一个 'mctb' 资源，则当您的应用程序调用 InitMenus时，菜单管理器可读该资源并将该信息存储在应用程序的菜单色彩信息表中。如果您提供一个具有相同的资源标识符的 'mctb' 资源作为 ' MENU' 资源，则当您使用 GetMenu 读菜单的资源说明（或使用GetNewBar读菜单栏中的所有菜单）时，菜单管理器也可以读任何相关的 'mctb' 资源（如果存在的话）。3 - 98 页的“菜单色彩信息表记录”中介绍了菜单色彩信息表的格式。

图 3-40 是经编译的 'mctb' 资源的格式。

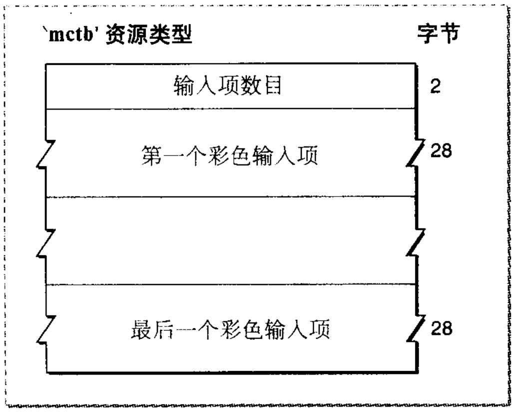


图 3-40　经编译的菜单色彩信息表（'mctb'）资源的结构

一个 'mctb' 资源的编译版本中包含下列元素：

- 菜单彩色输入项说明的编号计数
- 菜单彩色输入项的可变编号

一个彩色输入项可定义菜单和菜单栏各部分的彩色。图3 - 41 表示在一个 'mctb' 资源中经编译的菜单彩色输入项的格式。在 'mctb' 资源中的每个菜单彩色输入项包含下列内容：

- 菜单标识符表示此输入项是菜单命令项目输入项还是菜单标题输入项，0 表示此输入项是菜单栏输入项，－99 表示此输入项是该资源的最后一个输入项
- 菜单命令项目编号表示此输入项是一个菜单命令项目输入项，0 表示此输入项是一个菜单标题或是菜单栏输入项。菜单标识符和菜单命令项目共同确定如何说明菜单彩色输入项的类型。请参阅第3 - 100页表3 - 7中关于菜单标识符和菜单命令项目规范如何定义菜单彩色输入项类型的全面介绍
- RGB1：对于菜单栏输入项，它是菜单标题的系统设定色彩；对于菜单标题输入项，它是特定菜单的标题彩色；对于菜单命令项目输入项，是特定项目的标记彩色
- RGB2：对于菜单栏输入项，它是被显示菜单的系统设定背景色；对于菜单标题输入项，它是菜单栏的系统设定色彩；对于菜单命令项目输入项，它是特定项目文字的彩色

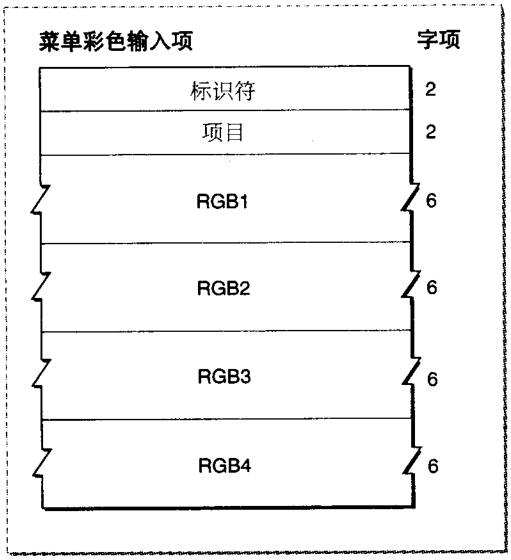


图 3-41　在 'mctb' 资源中一个菜单彩色输入项的结构

- RGB3: 对于菜单栏输入项，它是被显示菜单中项目的系统设定色彩；对于菜单标题输入项，它是特定菜单中项目的系统设定色彩；对于菜单命令项目输入项，它是特定项目的等效命令组合键的彩色

- RGB4: 对于菜单栏输入项，它是菜单栏的系统设定色彩；对于菜单标题输入项，它是特定菜单的背景色；对于菜单命令项目输入项，它是特定菜单的背景色

菜单定义过程资源

如果您提供自己的菜单定义过程，则应将它存储在 'MDEF' 类型的资源中。您需提供菜单定义过程中经编译或经汇编的代码作为资源数据。您的过程的输入项点必须在该资源数据的开始处。

如果您在 'MENU' 资源中定义菜单（并使用 GetMenu 或 GetNewMBar 函数），您可指定菜单定义过程，菜单管理器使用它来管理 'MENU' 资源中的菜单。如果您使用 NewMenu 函数（而不是 'MENU' 资源），您的应用程序必须直接用所希望的菜单定义过程间接指针取代 menuProc 域中的标准菜单定义过程间接指针。

菜单管理器小结

Pascal 小结

常数

```
CONST
    noMark          = 0;      {menu item doesn't have a marking character}

    {values for the message parameter to the menu definition procedure}
    mDrawMsg        = 0;      {draw the menu items of a menu}
    mChooseMsg      = 1;      {highlight or unhighlight a menu item as }
                              { appropriate if the cursor is in a menu item}
    mSizeMsg        = 2;      {calculate the dimensions of a menu}
    mPopUpMsg       = 3;      {calculate the open pop-up box rectangle}

    textMenuProc    = 0;      {resource ID of standard menu definition }
                              { procedure}
    hMeuCmd         = 27;     {constant ($1B) specified as keyboard equivalent }
                              { to indicate a menu item has a submenu}
    hierMenu        = -1;     {constant used with InsertMenu routine to insert }
                              { a submenu or pop-up menu into the submenu }
                              { portion of the current menu list}
    mctAllItems     = -98;    {search for all items with the given ID}
    mctLastIDIndic= -99;      {last menu color table entry has this value }
                              { in the ID field of the entry}
```

数据类型

```
TYPE
    MenuInfo =                          {menu record}
    RECORD
        menuID:       Integer;          {number that identifies the menu}
        menuWidth:    Integer;          {width (in pixels) of the menu}
        menuHeight:   Integer;          {height (in pixels) of the menu}
        menuProc:     Handle;           {menu definition procedure}
        enableFlags:  LongInt;          {indicates whether menu and }
                                        { menu items are enabled}
```

```
        menuData:      Str255;          {title of menu}
        {itemDefinitions}               {variable-length data that }
                                        { defines the menu items}
    END;

    MenuPtr         = ^MenuInfo;        {pointer to a menu record}
    MenuHandle      = ^MenuPtr;         {handle to a menu record}
    MCEntry =                           {menu color entry record}
    RECORD
        mctID:         Integer;         {menu ID or 0 for menu bar}
        mctItem:       Integer;         {menu item number or 0 for }
                                        { menu title}
        mctRGB1:       RGBColor;        {usage depends on mctID and }
                                        { mctItem}
        mctRGB2:       RGBColor;        {usage depends on mctID and }
                                        { mctItem}
        mctRGB3:       RGBColor;        {usage depends on mctID and }
                                        { mctItem}
        mctRGB4:       RGBColor;        {usage depends on mctID and }
                                        { mctItem}
        mctReserved:   Integer;         {reserved}
    END;

    MCEntryPtr      = ^MCEntry;         {pointer to a menu color entry record}

    MCTable         = ARRAY[0..0] OF MCEntry;{menu color table}
    MCTablePtr      = ^MCTable;              {pointer to a menu color table}
    MCTableHandle   = ^MCTablePtr;           {handle to a menu color table}
```

菜单管理器例程

初始化菜单管理器

```
PROCEDURE InitMenus;
PROCEDURE InitProcMenu      (resID: Integer);
```

创建菜单

```
FUNCTION NewMenu            (menuID: Integer; menuTitle: Str255)
                             : MenuHandle;
FUNCTION GetMenu            (resourceID: Integer): MenuHandle;
```

从当前菜单列表中添加和取消菜单

```
PROCEDURE InsertMenu        (theMenu: MenuHandle; beforeID: Integer);
PROCEDURE DeleteMenu        (menuID: Integer);
PROCEDURE ClearMenuBar;
```

从一个`MBAR'资源中获取菜单栏说明

```
FUNCTION GetNewMBar         (menuBarID: Integer): Handle;
```

获取和设定菜单栏

```
FUNCTION GetMenuBar: Handle;
PROCEDURE SetMenuBar        (menuList: Handle);
FUNCTION GetMBarHeight: Integer;
```

绘制菜单栏

```
PROCEDURE DrawMenuBar;
PROCEDURE InvalMenuBar;
```

响应用户选择的菜单命令

```
FUNCTION MenuSelect         (startPt: Point): LongInt;
FUNCTION MenuKey            (ch: Char): LongInt;
FUNCTION MenuChoice: LongInt;
PROCEDURE HiliteMenu        (menuID: Integer);
FUNCTION PopUpMenuSelect    (menu: MenuHandle;
                            Top: Integer; Left: Integer;
                            PopUpItem: Integer): LongInt;
PROCEDURE SystemMenu        (menuResult: LongInt);
FUNCTION SystemEdit         (editCmd: Integer): Boolean;
```

获取用于菜单记录的间接指针

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMenuHandle      (menuID: Integer): MenuHandle;
FUNCTION HMGetHelpMenuHandle
                            (VAR mh: MenuHandle): OSErr;
```

添加和删除菜单命令项目

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE AppendMenu        (menu: MenuHandle; data: Str255);
PROCEDURE InsertMenuItem    (theMenu: MenuHandle; itemString: Str255;
                            afterItem: Integer);
```

```
PROCEDURE DeleteMenuItem    (theMenu: MenuHandle; item: Integer);
PROCEDURE AppendResMenu     (theMenu: MenuHandle; theType: ResType);
PROCEDURE InsertResMenu     (theMenu: MenuHandle; theType: ResType;
                            afterItem: Integer);
```

获取和设定菜单外观

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE EnableItem        (theMenu: MenuHandle; item: Integer);
PROCEDURE DisableItem       (theMenu: MenuHandle; item: Integer);
PROCEDURE GetMenuItemText   (theMenu: MenuHandle; item: Integer;
                             VAR itemString: Str255);
PROCEDURE SetMenuItemText   (theMenu: MenuHandle; item: Integer;
                             itemString: Str255);
PROCEDURE GetItemStyle      (theMenu: MenuHandle; item: Integer;
                             VAR chStyle: Style);
PROCEDURE SetItemStyle      (theMenu: MenuHandle; item: Integer;
                             chStyle: Style);
PROCEDURE GetItemMark       (theMenu: MenuHandle; item: Integer;
                             VAR markChar: Char);
PROCEDURE SetItemMark       (theMenu: MenuHandle; item: Integer;
                             markChar: Char);
PROCEDURE CheckItem         (theMenu: MenuHandle; item: Integer;
                             checked: Boolean);
PROCEDURE GetItemIcon       (theMenu: MenuHandle; item: Integer;
                             VAR iconIndex: Byte);
PROCEDURE SetItemIcon       (theMenu: MenuHandle; item: Integer;
                             iconIndex: Byte);
PROCEDURE GetItemCmd        (theMenu: MenuHandle; item: Integer;
                             VAR cmdChar: CHAR);
PROCEDURE SetItemCmd        (theMenu: MenuHandle; item: Integer;
                            cmdChar: CHAR);
```

排列菜单

```
PROCEDURE DisposeMenu       (theMenu: MenuHandle);
```

计算菜单中项目的数目

```
FUNCTION CountMItems        (theMenu: MenuHandle): Integer;
```

醒目显示菜单栏

```
PROCEDURE FlashMenuBar      (menuID: Integer);
PROCEDURE SetMenuFlash      (count: Integer);
```

重新计算菜单尺寸

```
PROCEDURE CalcMenuSize      (theMenu: MenuHandle);
```

管理菜单色彩信息表中的输入项

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMCInfo: MCTableHandle;
PROCEDURE SetMCInfo         (menuCTbl: MCTableHandle);
PROCEDURE DisposeMCInfo     (menuCTbl: MCTableHandle);
FUNCTION GetMCEntry         (menuID    : Integer; menuItem: Integer)
                             : MCEntryPtr;
PROCEDURE SetMCEntries      (numEntries: Integer;
                             menuCEntries: MCTablePtr);
PROCEDURE DeleteMCEntries   (menuID: Integer; menuItem: Integer);
```

应用程序定义的例程

```
PROCEDURE MyMenuDef         (message: Integer; theMenu: MenuHandle;
                             VAR menuRect: Rect; hitPt: Point;
                             VAR whichItem: Integer);
```

C 小结

常数

```
enum {
    #define noMark     '\0' /*menu item doesn't have a marking character*/

    /*values for the message parameter to the menu definition procedure*/
    mDrawMsg       = 0,     /*draw the menu items of a menu*/
    mChooseMsg     = 1,     /*highlight or unhighlight a menu item as */
                            /* appropriate if the cursor is in a menu item*/
    mSizeMsg       = 2,     /*calculate the dimensions of a menu*/
    mPopUpMsg      = 3,     /*calculate the open pop-up box rectangle*/

    textMenuProc   = 0,     /*resource ID of standard menu definition */
                            /* procedure*/
    hMenuCmd       = 27,    /*constant ($1B) specified as keyboard */
                            /* equivalent to indicate an item has a submenu*/
    hierMenu       = -1,    /*constant used with InsertMenu to insert */
                            /* a submenu or pop-up menu into the submenu */
                            /* portion of the current menu list*/
```

```
    mctAllItems    = -98,     /*search for all items with the given ID*/
    mctLastIDIndic= -99       /*last menu color table entry has this value */
                              /* in the ID field of the entry*/
};
```

数据类型

```
struct MenuInfo {                       /*menu record*/
        short     menuID;               /*number that identifies the menu*/
        short     menuWidth;            /*width (in pixels) of the menu*/
        short     menuHeight;           /*height (in pixels) of the menu*/
        Handle    menuProc;             /*menu definition procedure*/
        long      enableFlags;          /*indicates whether menu and */
                                        /* menu items are enabled*/
        Str255    menuData;             /*title of menu*/
        /*itemDefinitions*/             /*variable-length data that */
                                        /* defines the menu items*/
    };

typedef struct MenuInfo MenuInfo;          /*pointer to a menu record*/
typedef MenuInfo *MenuPtr, **MenuHandle; /*handle to a menu record*/

struct MCEntry {                   /*menu color entry record*/
        short     mctID;                /*menu ID or 0 for menu bar*/
        short     mctItem;              /*menu item number or 0 for */
                                        /* menu title*/
        RGBColor mctRGB1;               /*usage depends on mctID and */
                                        /* mctItem*/
        RGBColor mctRGB2;               /*usage depends on mctID and */
                                        /* mctItem*/
        RGBColor mctRGB3;               /*usage depends on mctID and */
                                        /* mctItem*/
        RGBColor mctRGB4;               /*usage depends on mctID and */
                                        /* mctItem*/
        short     mctReserved;          /*reserved*/
    };

typedef struct MCEntry MCEntry;
typedef MCEntry *MCEntryPtr;      /*pointer to a menu color entry record*/
                                  /*menu color table*/
typedef MCEntry MCTable[1], *MCTablePtr, **MCTableHandle;
```

菜单管理器例程

初始化菜单管理器

```
pascal void InitMenus               (void);
pascal void InitProcMenu            (short resID);
```

创建菜单

```
pascal MenuHandle NewMenu           (short menuID, const Str255 menuTitle);
pascal MenuHandle GetMenu           (short resourceID);
```

添加菜单至当前菜单列表和从当前菜单列表中删除菜单

```
pascal void InsertMenu              (MenuHandle theMenu, short beforeID);
pascal void DeleteMenu              (short menuID);
pascal void ClearMenuBar            (void);
```

从`MBAR'资源中获取菜单栏说明

```
pascal Handle GetNewMBar            (short menuBarID);
```

获取和设定菜单栏

```
pascal Handle GetMenuBar            (void);
pascal void SetMenuBar              (Handle menuList);
#define GetMBarHeight()             (* (short*) 0x0BAA)
```

绘制菜单栏

```
pascal void DrawMenuBar             (void);
pascal void InvalMenuBar            (void);
```

响应用户选择的菜单命令

```
pascal long MenuSelect              (Point startPt);
pascal long MenuKey                 (short ch);
pascal long MenuChoice              (void);
pascal void HiliteMenu              (short menuID);
pascal long PopUpMenuSelect         (MenuHandle menu, short top, short left,
                                     short popUpItem);
pascal void SystemMenu              (long menuResult);
pascal Boolean SystemEdit           (short editCmd);
```

获取菜单记录的间接指针

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MenuHandle GetMenuHandle
                              (short menuID);
pascal OSErr HMGetHelpMenuHandle
                              (MenuHandle *mh);
```

添加和删除菜单命令项目

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void AppendMenu       (MenuHandle menu, ConstStr255Param data);
pascal void InsertMenuItem (MenuHandle theMenu,
                              ConstStr255Param itemString,
                              short afterItem);
pascal void DeleteMenuItem (MenuHandle theMenu, short item);
pascal void AppendResMenu   (MenuHandle theMenu, ResType theType);
pascal void InsertResMenu   (MenuHandle theMenu, ResType theType,
                              short afterItem);
```

获取和设定菜单命令项目的外观

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void EnableItem      (MenuHandle theMenu, short item);
pascal void DisableItem     (MenuHandle theMenu, short item);
pascal void GetMenuItemText (MenuHandle theMenu, short item,
                              Str255 itemString);
pascal void SetMenuItemText (MenuHandle theMenu, short item,
                              ConstStr255Param itemString);
pascal void GetItemStyle    (MenuHandle theMenu, short item,
                              Style *chStyle);
pascal void SetItemStyle    (MenuHandle theMenu, short item, short chStyle);
pascal void GetItemMark     (MenuHandle theMenu, short item,
                              short *markChar);
pascal void SetItemMark     (MenuHandle theMenu, short item,
                              short markChar);
pascal void CheckItem       (MenuHandle theMenu, short item,
                              Boolean checked);
pascal void GetItemIcon     (MenuHandle theMenu, short item,
                              short *iconIndex);
pascal void SetItemIcon     (MenuHandle theMenu, short item,
                              short iconIndex);
```

```
pascal void GetItemCmd      (MenuHandle theMenu, short item, short
                             *cmdChar);
pascal void SetItemCmd      (MenuHandle theMenu, short item, short cmdChar);
```

排列菜单

```
pascal void DisposeMenu     (MenuHandle theMenu);
```

计算菜单中的项目

```
pascal short CountMItems    (MenuHandle theMenu);
```

醒目显示菜单栏

```
pascal void FlashMenuBar    (short menuID);
pascal void SetMenuFlash    (short count);
```

重新计算菜单的尺寸

```
pascal void CalcMenuSize    (MenuHandle theMenu);
```

管理菜单色彩信息表的输入项

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MCTableHandle GetMCInfo(void);
pascal void SetMCInfo          (MCTableHandle menuCTbl);
pascal void DisposeMCInfo      (MCTableHandle menuCTbl);
pascal MCEntryPtr GetMCEntry   (short menuID, short menuItem);
pascal void SetMCEntries       (short numEntries, MCTablePtr menuCEntries);
pascal void DeleteMCEntries    (short menuID, short menuItem);
```

应用程序定义的例程

```
pascal void MyMenuDef     (short message, MenuHandle theMenu,
                           Rect *menuRect, Point hitPt,
                           short *whichItem);
```

汇编语言小结

数据结构

菜单信息数据结构

0	menuID	字	标识菜单的号码
2	menuWidth	字	菜单的宽度（以像素为单位）
4	menuHight	字	菜单的高度（以像素为单位）
6	menuDefHandle	长整型	菜单定义过程
10	menuEnable	长整型	启用标志
14	menuData	256字节	菜单标题，后面是菜单命令项目信息

全局变量

AtMenuBottom	在可滚动菜单底部的像素值。
MBarEnable	如果当前菜单栏中的所有菜单属于一个应用程序则该变量包含0 如果所有菜单属于一个桌面附件，则该变量包含一个非零值。
MBarHeight	包含菜单栏的当前高度，以像素为单位。
MBarHook	例程地址，当鼠标按钮按下时，MenuSelect 重复地调用该例程。
MenuCInfo	包含应用程序的菜单色彩信息表间接指针。
MenuDisable	包含最后所选项目的菜单标识符和项目号码，而不管该项目是否被禁止或有效。
MenuFlash	当用户选定一个菜单命令项目时，该菜单命令项目闪烁的当前计数（次数）。
MenuHook	例程地址，在醒目显示菜单标题和计算了菜单矩形之后，绘制菜单之前，MenuSelect 调用该例程。
TheMenu	包含在菜单栏中醒目显示的菜单的菜单标识符。
TopMenuItem	在可滚动菜单顶部的像素值。

结果代码

noErr	0	没错
paramErr	-50	参数列表出错
menuFullErr	-108	在堆区域中没有足够的空间
resNotFound	-192	不能读资源
hmHelpManagerNotInited	-885	没有设置帮助菜单

第四章　窗口管理器

介绍窗口 4–4
　当前活动窗口和非当前活动窗口 4–6
　窗口类型 4–8
　窗口区域 4–12
　对话框和警告栏 4–13
　控制部件 4–14
　桌面环境中的窗口 4–15
关于窗口管理器 4–16
　图形端口 4–17
　窗口记录 4–19
　彩色窗口 4–20
　窗口中的事件 4–21
使用窗口管理器 4–22
　管理多个窗口 4–23
　创建一个窗口 4–25
　　定义窗口资源 4–25
　　凭借资源创建一个窗口 4–27
　　在桌面环境中对文稿窗口定位 4–30
　绘制窗口内容 4–39
　更新内容区域 4–40
　保留更新区域 4–41
　处理窗口中的事件 4–41
　　处理窗口中的鼠标事件 4–42
　　处理窗口中的键盘事件 4–47
　　处理更新事件 4–48
　　处理激活事件 4–50
　移动窗口 4–53
　缩放窗口 4–53
　调整窗口大小 4–57
　关闭窗口 4–60

隐藏和显示窗口	4-62
窗口管理器参考	4-64
数据结构	4-65
彩色窗口记录	4-65
窗口记录	4-69
窗口状态数据记录	4-70
窗口彩色表记录	4-71
辅助窗口记录	4-73
窗口列表	4-74
窗口管理器例程	4-74
初始化窗口管理器	4-74
创建窗口	4-75
命名窗口	4-85
显示窗口	4-86
检索窗口信息	4-91
移动窗口	4-94
调整窗口大小	4-99
缩放窗口	4-101
关闭窗口并取消了配给窗口的存储器	4-103
保留更新区域	4-106
设定和检索其它窗口特性	4-109
操纵桌面环境	4-112
操纵窗口彩色信息	4-114
低级例程	4-116
应用程序定义的例程	4-120
窗口定义函数	4-120
资源	4-124
窗口资源	4-124
窗口定义函数资源	4-127
窗口色彩表资源	4-127
窗口管理器总结	4-129
Pascal 概述	4-129
常数	4-129
数据类型	4-131
窗口管理器例程	4-133
应用程序定义例程	4-135
C 概述	4-136
常数	4-136
数据类型	4-138
窗口管理器例程	4-139
应用程序定义的例程	4-142
汇编语言概述	4-143
数据类型	4-143
全局变量	4-144

第四章

窗口管理器

本章介绍您的应用程序如何使用窗口管理器来创建和管理窗口。

从提供和应答警告栏这样的个人机对话到创建和编辑文稿这样的打开－结束人机对话，Macintosh应用程序在大多数与用户的通信活动中都要使用到窗口。用户通常在屏幕的窗口中键入单词和公式，绘制图形，或输入数据。您的应用程序一般让用户把这种数据存储在文件中，打开存储的文件，并在窗口中显示存储的数据。请参阅《Inside Macintosh文件》(*Inside Macintosh: File*) 中的“文件管理介绍”一章，了解关于处理文件的详细情况。

窗口可以是任意大小或形状的，在可用内存空间允许的范围内，用户可以在屏幕上一次显示任意数目的窗口。

窗口管理器定义了一组标准窗口并提供了一组例程来管理这些窗口。窗口管理器可以帮助您的应用程序显示那些与Macintosh用户界面一致的窗口。请参阅《Macintosh用户界面设计原则》(*Macintosh Human Interface Guidelines*) 中关于窗口及其行为的详细说明。

您一般在资源中存储关于您窗口的信息。本章介绍标准的窗口资源。若要了解关于资源的一般信息，请参阅本书“Macintosh工具箱引论”一章。若要了解关于资源管理例程的信息，请参阅《Inside Macintosh工具箱扩展》(*Inside Macintosh： More Macintosh Toolbox*) 的“资源管理器”一章。

窗口管理器自身需要依赖QuickDraw，QuickDraw是Macintosh系统软件的一部分，可以用来处理图形的快速操纵。QuickDraw支持在图形端口中绘图，这些端口具有独立的坐标系统，是个独立而全面的绘图环境。每个窗口代表一个图形端口，在《Inside Macintosh绘图》(*Inside Macintosh： Imaging*) 中有相关的说明。

您的应用程序若要保留它的窗口，它需要知道用户正在桌面环境中采取什么行动。您的应用程序通过事件接收这种信息，这些事件能够说明用户的动作并报告关于您的应用程序处理状态的情况。本章介绍影响窗口显示的事件，并把按下鼠标和键盘事件看做与窗口操作有关。若要了解关于事件和您的应用程序怎样处理这些事件的详细说明，请参阅本书“事件管理器”一章。

大多数文稿窗口都包含控制部件，这些控制部件都是屏幕图像，用户通过操纵这些图像来控制应用程序的显示和执行。本章介绍了在窗口中用得最多的一些控制部件。若要进一步了解关于创建和响应控制部件的情况，请参阅本书“控制部件管理器”一章。

当用户创建新文稿或打开已存在的文稿时，您可以使用窗口管理器来创建和显示文稿窗口。当光标在您的应用程序创建的窗口中，用户点按或按住鼠标按钮时，您可以使用窗口管理器来确定鼠标动作的位置并相应地更改窗口内容的显示。当用户关闭窗口时，您可以使用窗口管理器从屏幕中取消该窗口。

本章介绍窗口管理器怎样支持窗口，然后解释您应如何使用窗口管理器来完成下列任务:

- 创建和显示窗口
- 处理窗口中的事件
- 当用户移动或重新调整窗口大小时更改窗口的显示
- 取消窗口

介绍窗口

窗口是一种用户界面单元，屏幕上的一个区域，用户可以在其中输入或显示信息。

用户可以同时在桌面环境中拥有源自许多不同应用程序的多个窗口。用户可以更改大多数窗口的大小和位置，可以将窗口全部或部分地置于其它窗口的前面。图4－1给出了桌面环境中的几个窗口。

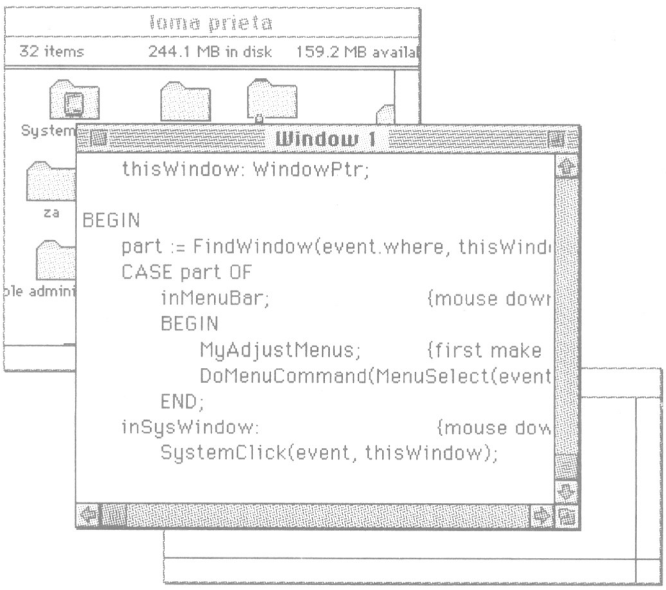


图 4-1 桌面环境中的多个窗口

您的应用程序一般创建文稿窗口，允许用户输入和显示文字、图形或其它信息。若想了解有关全彩色文稿窗口的实例，请参阅本书开头的调色板。

通过文稿窗口可以对文稿进行显示－－如果文稿比窗口大,则可以通过窗口是文稿的一部分,并通过窗口滚动条浏览整个文稿内容。您的应用程序可以在屏幕上显示一个或多个窗口，每个窗口显示一个文稿或显示处理文稿的辅助信息。

窗口管理器可以定义和支持一组标准窗口元素,用户可以通过它们来操纵窗口。您的应用程序应该遵守关于绘制窗口，移动窗口、调整窗口大小和关闭窗口的标准规则，这一点十分重要。通过提供标准的界面，您使有经验的用户能马上熟悉您应用程序的许多方面，允许他们把注意力集中于学习它的独特功能。

图4－2举例说明了标准的文稿窗口和它的各单个元素。

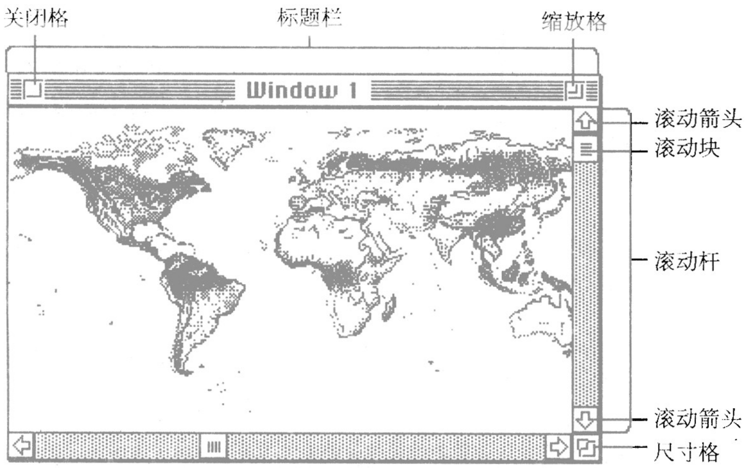


图4-2　文稿窗口

标题栏的作用是显示窗口的名称并指出该窗口是否是当前活动窗口。窗口管理器在标题栏的中心用系统设定的字体大小显示窗口的标题。如果系统字体在罗马语系系统中，则标题栏为20个像素高。

当用户创建一个新文稿时，您通常显示一个新的文稿窗口，其标题用小写字母拼写，称作“未标题文稿 1”。如果用户没有存储第一个窗口就创建了第二个文稿窗口，您就可以将第二个窗口的标题设定为“未标题文稿 2”，在单词与号码之间应有一个空格。只要用户继续创建新窗口而不存储前面已编号的未标题窗口，您就继续在标题的号码中加1。

当用户打开一个存储的文稿时，您应把文稿的名称指派给它所在的窗口。

用户希望通过拖拽窗口的标题栏来移动窗口。您可以通过调用窗口管理器的DragWindow过程来移动窗口，4-53页的“移动窗口”中有相关的说明。

关闭格为用户提供了一种快速关闭窗口的方法。您可以使用 TrackGoAway 函数在关闭格中跟踪鼠标活动，用 CloseWindow 和 DisposeWindow 过程来关闭窗口。4-60 页的“关闭窗口”中有关于关闭窗口的说明。

缩放格为用户提供了一种在两个不同窗口大小间快速切换的方法。您可以使用 TrackBox 函数在缩放格中跟踪鼠标活动，用 ZoomWindow 过程来缩放窗口。4 - 53 页的“缩放窗口”中有关于缩放窗口的介绍。

尺寸格允许用户更改窗口的大小和尺寸。使用GrowWindow函数在尺寸格中跟踪鼠标活动，用 SizeWindow 过程调整窗口大小。4-57 页的“调整窗口大小”中有关于调整窗口大小的说明。

滚动杆可以让用户看到文稿的不同部分，从而用户能够在窗口中看到更多的信息。尽管 Macintosh 用户界面准则指定您把滚动杆放置在窗口的右下边缘，但滚动杆并不是窗口结构的一部分。您可以通过控制管理器创建和控制滚动杆，请参阅本书“控制管理器”一章中的有关说明。

内容区域是窗口的一部分，您的应用程序可以在内容区域显示文稿的内容、尺寸格和窗口控制部件。

窗口框架是窗口的一部分，由窗口管理器自动绘制，由标题栏，包括关闭格和缩放格和窗口的轮廓线组成。

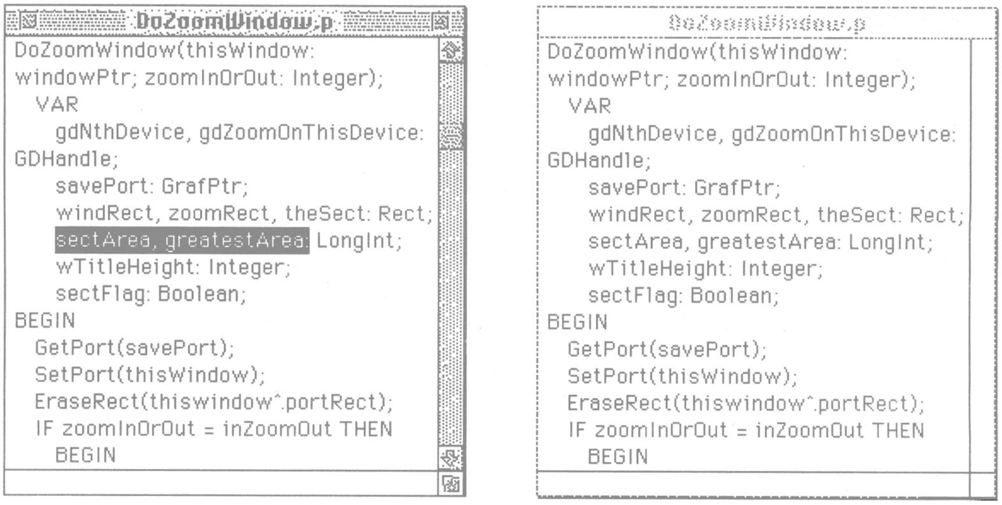
结构区域是指窗口所占据的整个屏幕区域，包括框架和内容区域（请参阅4-12页的图 4-10）。

当前活动窗口和非当前活动窗口

用户当前正在工作的窗口是当前活动窗口。当前活动窗口是桌面环境中最前面的窗口。它由可视的标题栏跑道线标识。

当前活动窗口是键盘活动的目标。它经常包含一个闪烁的插入点，用来指示新的文字或图形将出现的地方。当用户在当前活动窗口中选定文字时，您的应用程序应该用反白显示这些文字；如果该窗口变为非当前活动窗口，您应取消醒目显示。您可以使用第二类选定项技术，例如轮廓线，在失活窗口中标记选定项。您只应在当前活动窗口中显示滚动杆。图 4 - 3 给出了在当前活动和失活状态下的文稿窗口实例。

除了当前活动窗口以外，在桌面环境中所有的文稿窗口，不管它们是属于您的应用程序还是属于另外的应用程序，都是非当前活动窗口。您的应用程序可以处理非当前活动窗口中的文稿，但只有当前活动窗口能与用户对话。例如，如果用户在文件菜单中选择“存储”，您的应用程序仅存储当前活动窗口中的文稿。



当前活动文稿窗口　　　　失活的文稿窗口

图 4-3　当前活动和失活文稿窗口

如果希望一个窗口变成当前活动，那么用户在窗口内容或框架的任意位置点按即可实现。当用户激活了您的一个窗口时，您应调用窗口管理器来醒目显示窗口框架和标题栏；同时还应激活控制部件和窗口内容。一旦窗口变成当前活动，它会朝前移动，处在所有其它窗口的前面，呈现在用户面前。

当用户在一失活的文稿窗口中点按时，您应该使该窗口变成当前活动文稿窗口，但不要使该窗口中的任何选择项响应点按事件。若要在窗口中选定一个选择项，用户必须再次点按。这种行为可以防止用户在激活一个窗口时无意识地丢失数据。

附　注

Finder 在响应非当前活动窗口中的第一个点按事件时就会作出选择，因为这种动作更适合于 Finder 的使用方式。在应用程序创建的一些其它特殊用途窗口中，有时用户希望第一个点按就能导致一项选定。在文稿窗口中这种行为是不太合适的。◆

当属于您的应用程序的窗口变成非当前活动窗口时，窗口管理器就会刷新框架，取消标题栏的醒目显示并隐藏关闭和缩放格。您的应用程序隐藏控制制动能和尺寸格并取消应用程序控制的单元的醒目显示。

当用户重新激活窗口时，便可以让窗口恢复它在停用前的状态，即在同样的位置绘制滚动杆，还原插入点或醒目显示以前的选定项。

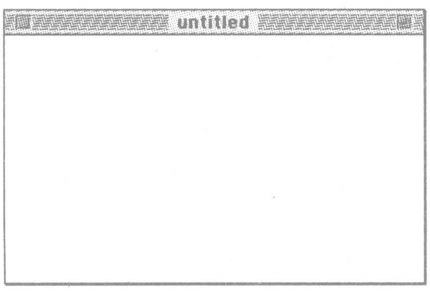
窗口类型

窗口的用途很多，外观也各不相同。窗口管理定义了许多窗口类型，可以满足大多数应用程序的基本需要。窗口类型是关于窗口的外观和它是如何工作的总体说明。一些窗口有标题栏而另一些则没有，窗口可以有如下窗口操纵元素的任意组合：关闭格、缩放格和尺寸格。

本节介绍了窗口管理器支持的九种基本的窗口类型以及它们的用途。通过指定如下这些窗口常数之一：ZoomDoCProc、 dBoxProc、altDBoxProc、plainDBoxProc、movableDBoxProc、noGrowDocProc、documentProt、ZooNoGrow 和 rDocProc，您可以创建这些类型的窗口。若想了解关于创建窗口的说明，请参阅4 － 25页的“创建窗口”。

若要为用户提供最大的灵活性和控制部件，可以把ZoomDocProc窗口类型用于您的文稿窗口。ZoomDocProc 窗口支持图 4 － 2 给出的所有的窗口操纵单元：标题栏、关闭格、缩放格和尺寸格。然而，窗口管理器并没有必要绘制关闭格和尺寸格。您必须调用窗口管理器的DrawGrowIcon过程来绘制尺寸格，当您创建窗口时您可以有选择地来绘制尺寸格，当您创建窗口时您可以有选择地隐藏关闭格。若想了解定义窗口特性的详细情况，请参阅 4 － 25 页的“创建窗口”。

图 4-4 给出一个 ZoomDocProc 类型的窗口实例，由窗口管理器绘制，带有关闭格，没有尺寸格和滚动杆，尺寸格和滚动杆可以由你随后添加。



zoomDocProc

图 4-4　一个 ZoomDocProc 类型的窗口

在大多数情况下，一个ZoomDocProc类型的窗口应该同时包含一个关闭格和一个尺寸格。当窗口容纳不下相关文稿包含的数据时，您可以通过滚动杆来调整文稿内容的显示。图 4 － 5 给出了一个带有尺寸格和滚动杆的 ZoomDocProc 类型的窗口。

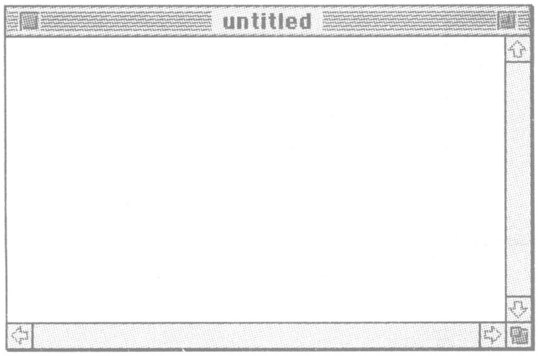
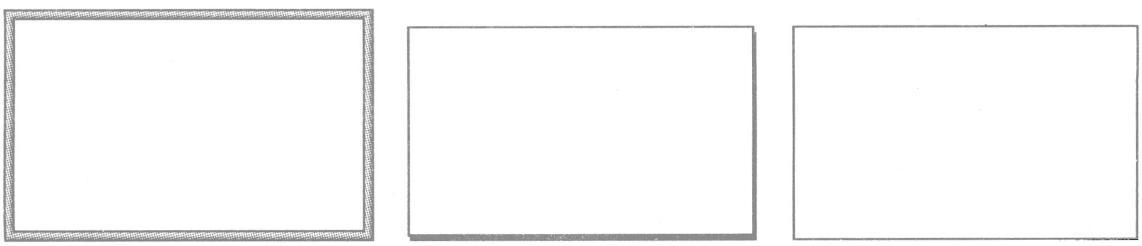


图 4-5　ZoomDocProc 类型的窗口，带有尺寸格和滚动杆

您也可以使用窗口来显示警告栏和对话框。本节介绍警告栏和对话框的窗口类型。若想了解关于不同种类的警告栏和对话框的更全面的信息，请参阅本书“对话管理器”一章。

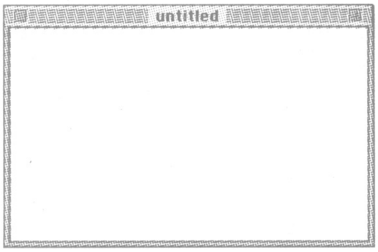
警告栏和位置固定的定模式对话框不包含窗口操纵单元。用户不能以手动方式对窗口进行移动，调整大小，或关闭等动作。警告栏或定模式对话框将保留在屏幕上作为当前活动窗口，直到对话管理器或您的应用程序取消它为止(通常是在用户点按按钮结束对话之时)。图 4 - 6 给出了可警告栏和固定位置定模式对话框的三种窗口类型。



dBoxProc　　altDBoxProc　　plainDBoxProc

图 4-6　警告栏和固定位置定模式对话框的窗口类型

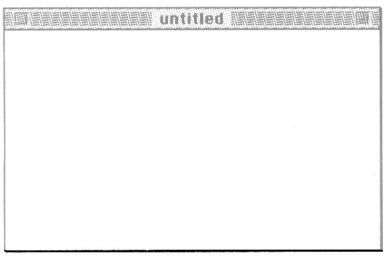
当您要让用户移动定模式对话框窗口以便显示可能被窗口遮住的文字时，
可移动的定模式对话框。可移动定模式对话框虽不能够对窗口调整大小、关闭或缩放，但可以移动窗口。下页的图 4 - 7 给出了 movableDBoxProc 窗口类型。像固定位置定模式对话框一样，可移动定模式对话框将保持当前活动状态直到用户完成对话为止。



movableDBoxProc

图 4-7　movableDBoxProc 类型的窗口

只要有可能,应避免使用定模式对话框而使用不定模式对话框,它允许用户在不取消对话框时仍能够执行其它任务。用于显示不定模式对话框的noGrowDocProc类型的窗口，可以移动或关闭，但不能调整大小或缩放。如果有必要，您可以采用其它窗口类型作为不定模式对话框，但如果您保持您的对话框尽可能的简单的话，就能很容易地遵守 Macintosh 用户界面准则。图 4 － 8 给出了不定模式对话框的窗口实例。

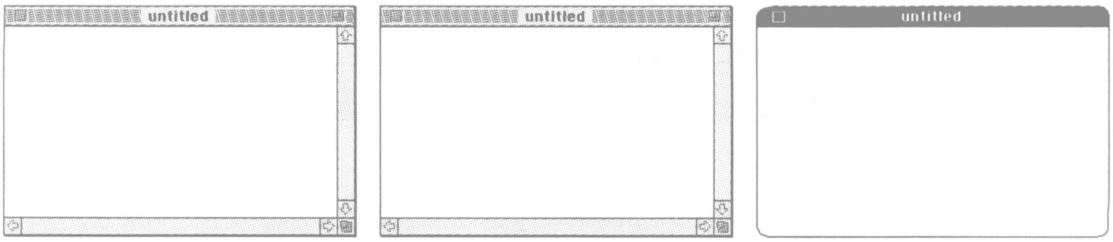


noGrowDocProc

图 4-8　noGrowDocProc 类型的窗口

窗口管理器也支持几个不常用的窗口类型。例如，documentProc 窗口类型，有一个标题栏并支持关闭格和尺寸格，但不支持缩放格。ZoomNoGrow 窗口类型实际上是永远也不合适的: ZoomNoGrow 支持关闭格和缩放格，但不支持尺寸格。rDocProc 窗口类型是一种圆角窗口，有标题栏和关闭格；桌面附件使用这种窗口。图 4 － 9 给出了这三种不常用的窗口类型。

窗口定义函数定义窗口的总体外观和特性。当系统软件和各种窗口管理器例程需要执行某些依赖于窗口的动作时，如绘图或调整窗口框架的大小，它们会调用窗口的窗口定义函数。



documentProc　　zoomNoGrow　　rDocProc

图 4-9　不常用的窗口类型

窗口管理器能够提供处理9种标准窗口类型的两个标准窗口定义函数。窗口定义函数绘制窗口的框架，绘制关闭格和窗口标题（如果有的话），确定光标处在窗口里的哪个区域，计算窗口的结构和内容区域，绘制窗口的缩放格（如果有的话），绘制窗口的尺寸格（如果有的话），并执行一切特殊的初始化或处理任务。

单个窗口定义函数可以支持多达16种不同的窗口类型。窗口定义函数为它支持的每个窗口类型定义一变量代码，这是一个从0至15的整数。

窗口定义标识符是一个综合了窗口的定义函数和它的变量代码的值。(窗口定义函数的资源标识符存储在整数的高12位，变量代码存储在低4位中。) 本节中介绍的窗口类型常数实际上就是窗口定义标识符。

常数	窗口定义标识符	说明
documentProc	0	可移动，可调整大小的窗口，没有缩放格
dBoxProc	1	警告栏或定模式对话框
plainDBox	2	单色框
altDBoxProc	3	带阴影的单色框
noGrowDocProc	4	可移动窗口，没有尺寸格或缩放格
movableDBoxProc	5	可移动的定模式对话框
ZoomDocProc	8	标准文稿窗口
ZoomNoGrow	12	可缩放，不能调整大小的窗口
rDocProc	16	圆角窗口

如果您需要具有不常见特性的窗口，您可以提供您自己的窗口定义函数，4-120页的窗口定义函数有相关的说明。一定要确保窗口行为遵守《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 中的准则。

窗口区域

窗口管理器可以识别许多特殊目的的窗口区域，这些区域是由窗口管理器或窗口定义函数定义的。

最明显的窗口区域是可视窗口的各部分，用户操纵这些部分来控制显示。这些窗口区域对应着标准窗口的各个部分。拖拽区域是由标题栏占据的区域，关闭格和缩放格除外。（用户可以通过拖拽窗口的标题栏来移动窗口。）大小区域，关闭区域和缩放区域是分别由尺寸格、关闭格和缩放格所占据的区域。

当光标处在您的一个窗口中时，用户按下了鼠标按钮，您可以调用窗口管理函数FindWindow来确定鼠标按下事件发生的区域。（FindWindow函数调用该窗口的窗口定义函数，窗口定义函数能够定义和解释窗口的操纵区域。）根据所得的结果，您应随后调用相应的窗口管理器例程或您自己的例程来处理事件。若想了解当用户按下鼠标按钮时如何确定光标位置的进一步介绍，请参阅4 － 42页的“处理窗口中的鼠标事件”。若想了解关于怎样使用窗口管理器例程来移动、调整大小、关闭和缩放窗口的讨论，请参阅4 － 53页的“移动窗口”和以后各节内容。

窗口管理器还能使得由它自动绘制的窗口部分和由您的应用程序绘制的部分之间存在明确的差别。窗口管理器绘制窗口框架－标题栏，包括关闭格和缩放格，以及窗口的轮廓线，（窗口管理器也绘制尺寸格，但仅当您的应用程序调用DrawGrowIcon过程时。）您的应用程序负责绘制内容区域－－文稿的内容、尺寸格和窗口控制部件（包括滚动杆）都在该窗口部分中显示。

一个窗口（包括窗口轮廓线、标题栏和内容区）占据的所有屏幕区域称为结构区域。图4 － 10给出了一个窗口的框架、内容区域和结构区域。



图4－10　窗口框架、内容区域和结构区

与一个窗口相关联的图形端口的绘图区域仅包括该窗口的内容区域。

当用户在桌面环境中创建、移动、调整大小和关闭窗口时，窗口可能会被部分地盖住和暴露。窗口管理器将跟踪这些变化，为每个窗口累积一个叫做更新区域的动态区域。更新区域包含窗口内容区域需要更新的所有区域。事件管理器可以周期性地扫描桌面环境中所有窗口的更新区域，为那些更新区域非空的窗口生成更新事件。当您的应用程序接收一个更新事件时，它将刷新更新区域。您的应用程序和窗口管理器都可以操纵窗口的更新区域。4 - 40 页的“更新内容区”和4 - 41 页的“保留更新区域”这两节说明了窗口管理器和您的应用程序是如何跟踪和使用更新区域的。

对话框和警告栏

Macintosh应用程序使用警告栏和对话框向用户传送消息并征求信息。例如，一个文字处理应用程序可能会显示警告栏告诉用户新插入的图形超出纸张的边界。它还可能显示一个对话框，用户可以在其中指定页边、制表标记以及其它格式信息（本书的“对话管理器”一章解释了如何使用各种警告栏和对话框）

警告栏和对话框仅仅是特殊用途的窗口。您可以通过对话管理器来处理所有的警告栏和大多数定模式对话框，对话管理器自身需调用窗口管理器。您可以为对话管理器提供警告栏和对话框中的项目列表，对话管理器显示窗口，告诉您用户正在操纵的项目，并当用户完成时消去窗口。您的应用程序提供用户在警告栏和对话框中选定项的代码。

尽管您可以为您的警告栏和定模式对话框指定任何窗口类型，处理警告栏和定模式对话框的对话管理器函数并不支持窗口操纵。因而您应将没有标题栏或尺寸格的窗口类型用于警告栏和定模式对话框，最典型的是dBoxProc窗口类型（当用户响应一个定模式对话框时，在对话框的菜单栏或内容区域之外的鼠标按下事件的结果仅产生系统示警声。注意，当 ModalDialog 过程正在处理事件时，进程管理器不执行主切换）

您可以把MovableDBox窗口类型用于可移动定模式对话框。您的应用程序可以使用对话管理器来帮助处理可移动定模式对话框中的事件，本书的“对话管理器”一章中有这方面的介绍。但您的应用程序必须处理窗口操纵事件－－通常仅是移动可移动定模式对话框窗口的事件。

把noGrowDocProc窗口类型用于不定模式对话框。您一般使用对话管理器来处理不定模式对话框中的事件，很像可移动定模式对话框中的事件。您的应用程序就像在文稿窗口中一样处理不定模式对话框中的窗口操纵事件。

如果您使用复杂的对话框，您会发当前活动窗口管理器以及工具箱的其它部分来创建和管理您自己的对话框窗口比用对话管理器更有效。若想了解评估对话框复杂性时要考虑的特性列表以及自定对话框的实例，请参阅本书“对话管理器”一章。

控制部件

大多数窗口都含有控制部件，它们是一些屏幕图像，用户可以通过操纵它们来控制您的应用程序的显示和执行。在文稿窗口中最常见的控制部件是滚动杆，图4 - 11给出了实例。

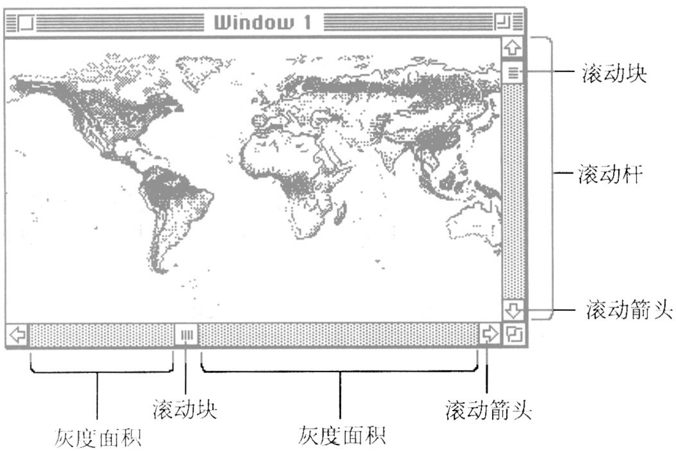


图4-11　滚动杆

您可以使用滚动杆来了解窗口中显示的文稿部分在整个文稿中所处的相对位置。您应该允许用户通过拖拽滚动块或单击灰色面积或滚动箭头将部分文稿移进或移出窗口。每当有在可用空间不能一次显示完的数据时，您就应激活窗口的滚动杆。

您可以使用控制管理器来创建、显示和操纵窗口中的滚动杆和其它任何控制部件。每个控制部件都“属于”一个窗口，并显示在代表该窗口的图形端口内。窗口管理器能够为您的应用程序创建的每个窗口保留一个控制列表，它是指向与窗口相关的控制部件说明的一个输入项系列。

大多数警告栏和对话框都包含按钮，按钮外观呈圆角矩形，当用鼠标单击时它们会导致一个立即的或连续的动作，大多数对话框还包含像选项按钮这样的辅助屏幕图像，它们显示并保留设定状态。图4-12给出了一个带有按钮、选项按钮和一些其它控制部件及对话项目的对话框。

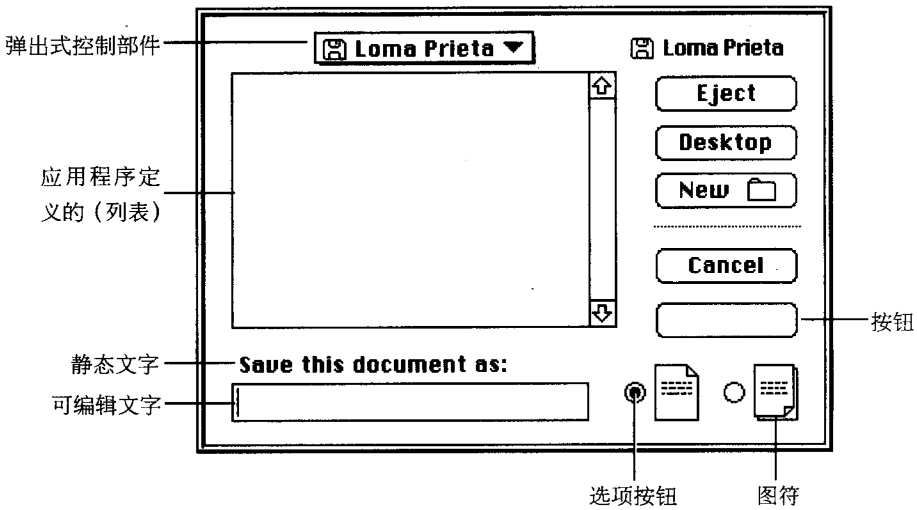


图4-12 对话框中的控制部件

按钮通常仅在警告栏和对话框中出现。图4-12中给出的其它大多数元素仅在对话框中出现。如果您使用对话管理器来创建您的警告栏和对话框，它将为您绘制控制部件并让您知道用户什么时候单击了一个控制部件。您自己可以调用控制管理器来显示和跟踪应用程序创建的任何窗口中的按钮和控制部件。您也可以编写自己的控制部件定义函数来创建并控制其它种类的控制部件。若想了解关于怎样创建和支持控制部件的全面介绍，请参阅本书“控制管理器”一章。

桌面环境中的窗口

来自不同应用程序的多个窗口可以同时在桌面环境中出现。窗口管理器使用它自己的专用数据结构窗口列表来跟踪所有窗口。输入项按照它们在桌面环境中的顺序出现在窗口列表中，从最前面的当前活动窗口开始。当用户更改桌面环境中窗口的顺序时，窗口管理器将生成事件，通知您的应用程序根据需要激活、失活和刷新窗口。窗口管理器防止您在其它应用程序的窗口中意外地绘图。

用户一次只能与一个应用程序对话。正与用户对话的应用程序 也就是用户正在其窗口中工作的应用程序）是当前活动应用程序，或前台进程，其它的则为后台应用程序，或后台进程。用户切换应用程序的一种方法是单击属于后台进程的窗口。进程管理后生成挂起事件，告诉前一个当前活动窗口它将被挂起，告诉最新的当前活动窗口它可以继续执行处理工作。若想了解关于前台和后台进程的工作，和支持同时运行多个应用程序运行的事件的更详细情况，请参阅本书“事件管理器”一章。）

您的应用程序很可能同时在桌面环境中拥有多个窗口：一个或多个文稿窗口，或许一个或多个对话框窗口，也许还有一些其它特殊用途的窗口。4-23 页的“管理多窗口”一节介绍了保持多窗口跟踪的技术。

在最初的Macintosh电脑上，桌面环境局限于一已知尺寸的单个屏幕。而现在的系统可支持各种大小和功能的多个显示器。若要把应用程序的窗口放置在桌面环境中合适的地方，您的应用程序必须注意哪个屏幕空间是可用的，用户正在哪里工作。若想了解管理窗口放置的规则，请参阅《Macintosh用户界面设计原则》(*Macintosh Human Interface Guidelines*)。若想了解在多屏幕上管理窗口的技术，请参阅 4-30 页的“在桌面环境中定位文稿窗口”。

整个桌面环境区域—也就是没被菜单栏占据的屏幕区域—叫作灰度区域。窗口管理器在GrayRgn全局变量中保留一个灰度区域指针；您可以通过窗口管理器函数GetGrayRgn来检索灰度区域指针。

关于窗口管理器

窗口管理器能够为创建、移动、调整大小和操纵窗口提供一整套例程。另外，它也提供低级支持，管理桌面环境中窗口的层次，当桌面环境变化会影响到应用程序的窗口时向您的应用程序发出警告。您的应用程序和窗口管理器互相配合为用户提供一个一致的窗口界面。

例如，当光标在窗口标题栏的拖拽区域时，用户按下鼠标按钮，您就可以调用DragWindow过程，它在屏幕内移动窗口的虚线轮廓线，以响应鼠标运动。当用户释放鼠标按钮时，DragWindow调用MoveWindow过程，由它在窗口的新位置刷新窗口。如果由于移动，属于您的应用程序失活窗口部分地或全部地暴露出来，窗口管理器将触发一个更新事件，告诉您的应用程序刷新暴露的区域。

类似地，如果用户在一个失活窗口中单击，您可以调用SelectWindow过程。SelectWindow调整窗口的醒目显示和位置并生成激活事件告诉您的应用程序要激活和失活的窗口。

窗口管理器能够为9种基本的窗口类型提供内在的支持，在始于4－8页的“窗口类型”中有关于这些类型的说明。当您使用这些窗口类型时，窗口管理器绘制窗口的框架，确定光标在窗口的什么区域，计算窗口的结构和内容区域，绘制窗口的尺寸格，绘制窗口的关闭格和缩放格，并执行任何特殊的初始化或处理任务。如果有必要，您可以编写您自己的窗口定义函数来处理其它窗口类型。

图形端口

每个窗口代表一个QuickDraw图形端口，它是具有自己的坐标系统的绘图环境。[请参阅《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*)一书中关于图形端口和坐标系统的全面说明。]当您创建一个窗口时，窗口管理器创建一个图形端口，窗口的内容在其中显示。

窗口在屏幕上的位置在全局坐标中定义，该坐标反映了全部潜在绘图空间。QuickDraw和ColorQuickDraw识别的坐标平面的原点在主屏幕的左上角，其正向X轴向右延伸，正向Y轴向下延伸。在QuickDraw中，水平偏移被标以h，垂直偏移被标以v。图4－13说明了QuickDraw全局坐标系统。

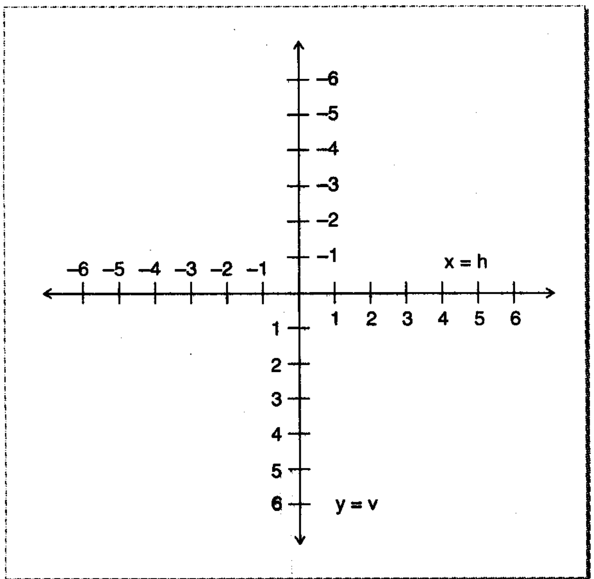


图4-13　QuickDraw全局坐标平面

附 注

对计算机图形来说，很方便的垂直轴定向与数学上的惯例不同。同时，坐标平面受QuickDraw坐标极限的限制，其范围是从 - 32,768至32,767。

QuickQraw在它自己的数据结构类型Point和Rect中储存点和矩形。在这些结构中，先是垂直坐标（v），后是水平坐标（h）。大多数，但不是所有的，处理点的QuickDraw例程要求您按这个顺序指定坐标。◆

当QuickDraw创建一个新的图形端口（通常是因为您已通过窗口管理器创建了一个新窗口）时，它在全局坐标中为端口定义一个边界矩形。边界矩形一般代表显示窗口的所有屏幕区域。边界矩形储存在图形端口数据结构中，在Color QuickDraw中它储存在叫做像素映像的结构的bounds域中，在QuickDraw中它储存在叫做位映像的结构的bounds域中。

图形端口数据结构也包括一个叫做protRect的域，它可以定义用来绘图的矩形。在代表窗口的图形端口中，portRect矩形代表窗口的内容区域。

附 注

当您在屏幕上设置一个窗口时，您应在全局坐标中指定它的内容区域的位置。记住要为窗口的标题栏留出空间。在主屏幕中，记住为菜单栏留出空间。在罗马语系系统中，标准的文稿标题栏和菜单栏都是20个像素高。您可以通过菜单管理器GetMBarHeight函数确定菜单栏的高度。您还可以通过比较窗口结构区域的顶部和窗口内容区域的顶部来计算标题栏的高度。请参阅4 - 55页的程序清单4 - 12中的过程实例，当在屏幕上放置一个窗口时它考虑菜单栏和标题栏。◆

在端口矩形内，绘图区域在局部坐标中说明—就是在端口矩形定义的坐标系统中。您在局部坐标中给窗口绘图，不需考虑窗口在屏幕上的位置（这在全局坐标中说明）。图4-14给出了一个窗口实例的局部和全局坐标系统，该窗口高180个像素，宽300个像素，它的内容区域距屏幕左上角的相对位置为：向下70个像素，向右60个像素。

当窗口管理器创建一个窗口时，它把局部坐标系统的原点置于窗口端口矩形的左上角。您可以通过QuickDraw过程SetOrigin重新定义端口矩形左上角的坐标。

事件管理器在全局坐标中说明鼠标事件，而且您可以在全局坐标中完成您的大部分窗口操纵。您一般需要在局部坐标中显示用户数据和操纵您的控制部件。当您需要在这两种坐标之间转换时，您可以调用QuickDraw函数GlobalToLocal和LocalToGlobal例程，在《Inside Macintosh绘图》(*Inside Macintosh : Imaging*) 中有相关的说明。

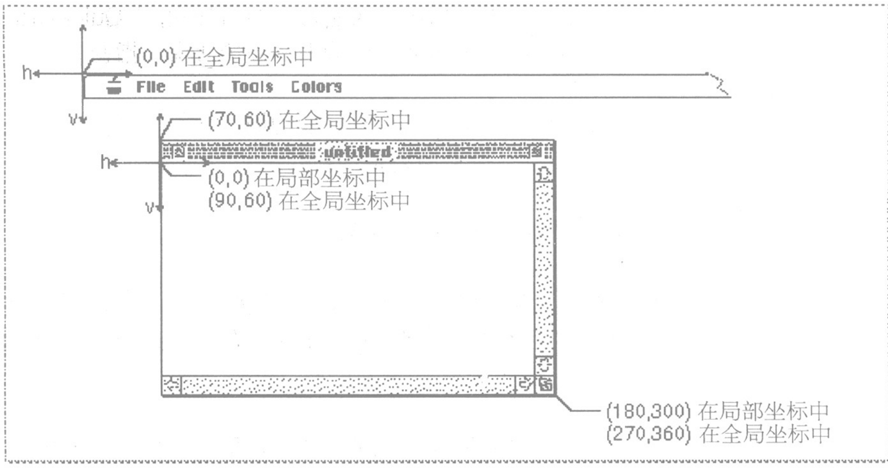


图 4-14　窗口的局部和全局坐标系统

窗口记录

每个窗口都有一些说明性特征，例如标题，控制部件列表和可视状态。窗口管理器在窗口记录中储存这些信息，窗口记录是一种 WindowRecord 类型的数据结构。

窗口记录包括

- 窗口的图形端口数据结构
- 窗口的分类，指定它是由窗口管理器直接创建还是由对话管理器间接创建
- 窗口标题
- 一系列标志，指定窗口是否可视，是否醒目显示，是否有缩放格，是否有关闭格
- 结构、内容和更新区域的指针
- 窗口定义函数的间接指针
- 窗口控制部件列表的间接指针
- 可选窗口内容图片的间接指针
- 您的应用程序需要时可使用的一个参考常数域

4 － 65 页的“彩色窗口记录”详细地说明了窗口记录。

窗口记录的第一个域是窗口的图形端口。因而 WindowPtr 数据类型被定义为图形端口的指针。

```
TYPE   WindowPtr   =   GrafPtr£ª
```

您可以通过在窗口的图形端口中绘图实现在窗口绘图，向QuickDraw绘图例程传递一个窗口指针。您也可以给大多数窗口管理例程传递窗口指针。

您通常并不需要存取或直接修改窗口记录的域。但当您这样做时，您可以通过窗口记录指针 WindowPeek 数据类型来表示它们。

```
TYPE WindowPeek= ^WindowRecord;
```

关闭格、拖拽格、缩放格和尺寸格并没有包括在窗口记录中，因为它们并没有必要针对在QuickDraw中定义的区域拥有正式的数据结构。窗口定义函数能够确定这些区域的所在位置。

您的应用程序很少直接存取窗口记录，当您对窗口作出任何改变时，如改变它的标题，窗口管理器会自动地更新窗口记录。窗口管理器也可以提供例程来修改和读取窗口记录的一些部分。

彩色窗口

自从推出了 Color QuickDraw 之后，窗口管理器已经能够支持彩色窗口。彩色窗口在彩色图形端口中显示，《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*) 对此加以说明。彩色窗口记录与4 - 19页的“窗口记录”中说明的窗口记录非常相似，区别仅在于它含有彩色图形端口而不是单色图形端口。

不管您的应用程序是否使用彩色显示，也不管当前是否安装了彩色显示器，只要 Color QuickDraw 可以使用，您的应用程序就应使用彩色窗口。一旦您已创建了一个窗口，您就可以将单色窗口的窗口记录和窗口指针用于彩色窗口的窗口记录和窗口指针，二者可以互换使用。

在一个设定为4位或更多彩色的显示器，窗口管理器能够自动地用彩色（或灰度，取决于显示器的功能）显示窗口的标题和部分框架以及控制部件。用户可以通过彩色控制板调整这些彩色。除一些罕见的情况外，您的应用程序不应该试图改变窗口框架的彩色。在设定显示 1 位彩色的显示器上，窗口管理器能够绘制黑白窗口标题、框架和控制部件。

窗口各元素的彩色由窗口彩色表控制，表中包含一系列的不同窗口元素部分代码和与每部分相关的 RGB 值。

因为用户可选定整个桌面环桌的窗口显示彩色，同时只要您不超越它，窗口管理器将自动地执行一些复杂的显示计算，所以您的应用程序一般使用系统设定的系统窗口彩色表。

如果您的应用程序显式控制窗口中的彩色，那么您也可以定义您自己的窗口彩色表。

您可以通过提供与窗口的'WIND'资源具有相同的资源标识符的窗口彩色表资源（即'wctb'类型的资源）为窗口定义窗口彩色表。窗口管理器在创建窗口记录的同时创建窗口彩色表。窗口管理器通过辅助窗口记录保留它自己的链接列表，辅助窗口记录可以使应用程序的窗口与它们相应的窗口彩色表联系起来。在4－71页的“窗口彩色表记录”和4－73页的“辅助窗口记录”中，有关于窗口彩色表和辅助窗口记录的说明。

除一些罕见的情况外，您的应用程序没有必要操纵窗口彩色表或辅助窗口记录。

为了与共享环境中的其它应用程序保持兼容，您的应用程序不应该直接操纵系统彩色表，而应使用调色板管理器，在《Inside Macintosh 绘图》（*Inside Macintosh : Imaging*）中有相关的说明。如果您的应用程序提供它自己的窗口和控制部件定义函数，这些函数应该像系统设定的定义函数一样为用户提供桌面环境彩色选定项。

窗口中的事件

事件是说明用户动作并报告您的应用程序处理状态的消息。窗口管理器生成两种事件：激活事件和更新事件。激活事件告诉您的应用程序一个指定窗口正变成当前活动或失活的窗口。更新事件告诉您的应用程序它必须刷新窗口内容区域的部分或全部。4－41页的“处理窗口中的事件”一节说明了这些事件什么时候发生以及您的应用程序如何响应。

窗口管理器的基本功能之一就是当您的应用程序接收到鼠标按下事件时报告光标所处的位置。窗口管理器函数 FindWindow 告诉您的应用程序光标是否在一窗口中，如果在窗口中，在哪个窗口，在该窗口的什么位置（也就是标题栏、拖拽区等等）。您可以把FindWindow函数当作鼠标按下事件的第一个过滤程序，把那些仅影响窗口显示的事件从操纵用户数据的事件中分离出来。

窗口管理器也提供一组例程帮助您履行标准的窗口操纵常规。

用户动作	应用程序响应
拖拽标题栏	移动窗口
拖拽尺寸格	调整窗口的大小
单击缩放格	在两种大小和位置间触发窗口，称作用户状态和标准状态。
单击关闭格	关闭窗口

下一节“使用窗口管理器”介绍了您可以如何使用窗口管理器来移动、调整大小和关闭窗口。

您可以调用控制管理器来处理窗口控制部件中的事件，在本书“控制管理器”一章中有相关的说明。如果您把对话管理器用于您的警告栏和定模式对话框，对话管理器将处理这些窗口中的键盘活动和鼠标事件。您也可以使用对话管理器来处理可移动定模式对话框和不定模式对话框的内容区域中的键盘活动和鼠标事件。但您的应用程序必须处理可移动定模式或不定模式对话框的标题栏和关闭格中的鼠标事件。

当您的应用程序呈当前活动用状态时，属于任何其它应用程序的窗口中的鼠标事件，包括 Finder，都将把您的应用程序切换到后台（除非有一个警告栏或一个定模式对话框正等待处理，在这种情况下对话管理器仅发出系统示警声）。

使用窗口管理器

实质上每个 Macintosh 应用程序都使用窗口管理器来简化窗口的显示和管理，以及检索关于用户活动的基本信息。

您的应用程序能够与窗口管理器合作为窗口提供标准的用户界面。例如当用户单击您的应用程序的一个失活窗口时，您可以调用过程 SelectWindow，它可以醒目显示新的当前活动窗口，取消前一个当前活动窗口的醒目显示，并生成触发两个受影响窗口的激活和失活的激活事件。

您的应用程序也可以使用窗口管理例程处理直接的窗口操纵。例如，如果当光标在窗口的标题栏中时用户按下鼠标按钮，您可以调用 DragWindow 过程来跟踪鼠标，在屏幕上拖拽窗口的轮廓线直到用户释放鼠标按钮为止。

您一般依据窗口资源创建窗口，这些资源类型为'WIND'。窗口管理器支持 9 种窗口类型，4 - 8 页的“窗口类型”中有关于这些窗口类型的说明。（您也可以编写您自己的窗口定义函数来支持您自己的窗口类型。窗口定义函数可以当作'WDEF'类型的资源储存。）警告栏窗口和对话框窗口使用警告（'ALRT'），对话（'DLOG'），和项目列表（'DITL'）资源；“对话管理器”一章介绍了怎样创建这些资源。大多数窗口包含控制部件，它由控制（'CNTL'）资源定义；“控制管理器”一章介绍了怎样创建控制资源。

通常您的应用程序在使用窗口管理器的同时，也使用控制管理器及对话管理器。您应使用控制管理器来定义、绘制并操纵窗口的控制部件。例如，若您的窗口包含滚动杆，您可以使用TrackControl函数在用户拖拽滚动杆时跟踪鼠标动作。您可以使用对话管理器来创建、显示并跟踪警告栏和对话框中的事件。

System 7为一个用标准窗口定义函数创建的窗口框架（即标题栏、缩放格和关闭格）提供了旁述。您应该为您窗口内容区域（即尺寸格、控制部件和数据区）以及您定义的任何窗口类型的窗口框架提供旁述。有关如何使用旁述的说明，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh : More Macintosh Toolbox*) 一书的“帮助管理器”一章。

在使用窗口管理器之前，您必须调用 InitGraf 过程对 QuickDraw 进行初始化，再调用 InitFont 过程对字体管理器进行初始化，最后调用 InitWindows 过程对窗口管理器进行初始化。

管理多个窗口

您的应用程序经常会在桌面环境中同时打开多个窗口：一个或多个文稿窗口，可能有一个或多个对话框，还可能有一些您专用的窗口。但是每次只能有一个窗口是现用的。

当您的应用程序接收到一个事件时，它根据当前活动窗口的种类和该事件发生的位置进行响应。当它在当前的文稿窗口的内容区域接收到一个鼠标按下事件时，您的应用程序就按照它自己的惯例进行：例如，插入文字、进行选定或添加图形。当它在菜单栏上收到一个鼠标按下事件时，您的应用程序就适当地启用或停用菜单命令项目—这也取决于当前的窗口类型及在该窗口中作的选定。例如，若用户在一个定模式对话框的可编辑文字域中有一个插入点，在编辑菜单中可能只有菜单命令项目“贴”可用，然后还需要在剪贴数据存储区中有可以贴的内容。

您可以使用各种策略对不同种类的窗口保持跟踪。窗口记录的refCon 域是专门留给应用程序使用的。您可以使用 refCon 域存储不同种类的数据，例如表示窗口类型的数值或说明窗口记录的间接指针。

本章的示例代码是从在全书中使用的SurfWriter应用程序中摘出的，它使用了一种混合的策略：

- 对于文稿窗口，refCon 域保持一个文稿记录的间接指针
- 对于不定模式对话框或可移动定模式对话框，refCon 域保持表示对话框类型的一个数值

您可能会发现其它更实用的方法。

SurfWriter 应用程序存储的文稿信息涉及到用户数据、窗口显示以及与文稿记录数据有关的文件（如果有的话）。文稿记录的形式如下:

```
TYPE MyDocRec =
  RECORD
     editRec:       TEHandle;      {handle to text being edited}
     vScrollBar:    ControlHandle; {control handle to the }
                                   { vertical scroll bars}
     hScrollBar:    ControlHandle; {control handle to the }
                                   { horizontal scroll bars}
     fileRefNum:    Integer;       {reference number for file}
     fileFSSpec:    FSSpec;        {FSSpec record for file}
     windowDirty:   Boolean;       {whether data has changed }
                                   { since last save}
     END;
     MyDocRecPt     = ^MyDocRec;
     MyDocRecHnd    = ^MyDocRecPtr;
```

SurfWriter 应用程序在它每次创建文稿窗口的同时，创建一个文稿记录，并且在窗口记录的 refCon 域中存储该文稿记录的间接指针。[有关如何使用文稿记录的完整示例，请参阅《Inside Macintosh 文件》(*Inside Macintosh : Files*) 中"文件管理介绍"一章内容。]

当 SurfWriter 创建一个不定模式对话框或可移动定模式对话框时，它存储一个代表该对话框的常数（即它可在对话资源中指定该常数，窗口管理器在创建窗口记录时就把 refCon 域设定为该值)。例如，一个值为 20 的 refCon 可能指定了一个接受查找命令输入项的不定模式对话框，值为 21 的 refCon 可能指定了一个接受拼写核对程序输入项的不定模式对话框。

当 SurfWriter 在它的一个窗口中接受某个事件通知时，它首先确定窗口的功能，然后适当地调度该事件。程序清单 4-1 说明了一个应用程序定义的例程 MyGetWindow-Type，它决定了窗口类型。

附　注:

MyGetWindowType函数可从一组应用程序定义的窗口类型中确定一种窗口类型，应用程序定义的窗口反映了应用程序创建的不同的窗口类型。这些窗口类型不同于由定义函数定义的标准窗口类型，它们用于确定窗口的外观及特征。若要查找一个窗口属于何种标准窗口类型，请调用窗口管理器函数 GetWVariant。

本章下面的示例代码调用 MyGetWindowType 函数作为其事件处理过程的一部分，该函数在 4-41 页的"处理窗口中的事件"一节中说明。

程序清单 4-1 定义窗口类型

```
FUNCTION MyGetWindowType (thisWindow: WindowPtr): Integer;
VAR
     myWindowType:      Integer;
BEGIN
  IF thisWindow <> NIL THEN
  BEGIN
    myWindowType := WindowPeek(thisWindow)^.windowKind;
    IF myWindowType < 0 THEN               {window belongs to }
      MyGetWindowType := kDAWindow         { a desk accessory}
    ELSE
      IF myWindowType = userKind THEN      {document window}
        MyGetWindowType := kMyDocWindow
         ELSE                              {dialog window}
             MyGetWindowType := GetWRefCon(window);
                                           {get dialog ID}
  END
  ELSE
     MyGetWindowType := kNil;
END;
```

请注意，MyGetWindowType将核对窗口是否属于一个桌面附件。这一步保证了与系统软件的旧版兼容。当您的应用程序在System 7下运行时，它应该只接收自己的窗口事件及在其（存储器）分区中启用的桌面附件窗口中的事件。有关（存储器）分区的说明请参阅《Inside Macintosh内存》(*Inside Macintosh : Memory*)，有关启用应用程序和启用桌面附件的说明请参阅《Inside Macintosh进程》(*Inside Macintosh: Process*)。

创建一个窗口

通常您在窗口（'WIND'）资源中指定窗口的特征，例如它们的初始大小、位置、标题和类型等。您一旦定义了窗口资源，就可以调用GetNewCWindow（或GetNewWindow）函数来创建窗口。

定义窗口资源

通常您对应用程序创建的每个窗口类型定义一个窗口资源。例如，如果您的应用程序同时创建了文稿窗口和专用窗口，您就可能希望为这两类窗口定义两个窗口资源。通过在窗口资源中定义窗口，您在用不同语言本地化您的窗口标题时，只需改变窗口资源。（您可用警告资源和对话资源指定警告栏和对话框的特征，这些内容在本书的“对话管理器”一章中说明。）

程序清单 4-2 以 Rez 输入格式给出了一个窗口资源，应用程序可以使用它来创建文稿窗口。该资源指定了从资源标识符为 128 的资源类型‘WIND’中创建的窗口的属性。系统软件在打开资源文件后立即将该资源装入存储器，而存储器管理器可以清除资源占据的存储器。

程序清单 4-2 文稿窗口的窗口（'WIND'）资源的 Rez 输入

```
#define rDocWindow     128
resource 'WIND' (rDocWindow, preload, purgeable) {
     {64, 60, 314, 460},     /*initial window size and location*/
     zoomDocProc,            /*window definition ID: */
                             /* incorporates definition function */
                             /* and variation code*/
     invisible,              /*window is initially invisible*/
     goAway,                 /*window has close box*/
     0x0,                    /*reference constant*/
     "untitled",             /*window title*/
     staggerParentWindowScreen
                             /*optional positioning specification*/
};
```

该资源第一个元素中的 4 个数值指定了窗口内容区域矩形的左上角和右下角的全局坐标，它们确定了内容区域的初始大小和位置。通过编程或本节后面介绍的一个可选的定位代码，您的应用程序可以在显示窗口前改变这个矩形。当在桌面环境中指定一个窗口的位置时，请记住为窗口框架留出空间，并在主屏幕上为菜单栏留出空间。

第二个元素包含了窗口定义标识符，它指定了用于处理窗口的窗口定义函数和一个可选的用于定义窗口类型的变量代码。如果您希望使用一个标准窗口类型（在 4-8 页的“窗口类型”中说明），您只需要指定一个窗口类型常数，它在 4-124 页的“窗口资源”中列出。

窗口资源的第三个元素，指定了窗口开始时是否可视。这个元素只确定窗口管理器在首次创建窗口时是否显示它，而不是窗口在屏幕上是否可见。(例如一个完全被其它窗口覆盖的窗口可能是“可见”的，尽管用户看不到它。) 通常您应以不可视状态创建新窗口，建立窗口内容区域，然后通过调用 ShowWindow 使它可视就能显示完整的窗口。

窗口资源的第四个元素指定了窗口是否有一个关闭格。只有某些标准窗口类型(zoomDocProc、noGrowDocProc、documentProc、zoomNoGrow 和 rDocProc)才能支持关闭格。若资源的第二域指定了一个不支持关闭格的窗口类型,关闭格元素就不起作用。窗口管理器在绘制窗口框架时绘制关闭格。

窗口资源的第五个元素是一个参考常数,您的应用程序可以在其中存储它需要的任何数据。当窗口管理器建立一个新的窗口记录时，它就在refCon域中存储您在这里指定的任何值。您也可以在这里放一个位置标志符（在这个例子中是 0X0)，然后您通过调用SetWRefCon 过程，自己设定refCon域。

窗口资源的第六个元素是一个用来指定窗口标题的字符串。

窗口资源的第七个元素是可选的,它指定一个新的定位规则,该规则优先于第一个元素中的矩形指定的窗口位置。在用于文稿窗口的窗口资源中，您通常可指定定位常数staggerParentWindowScreen。有关定位常数的完整列表及其作用，请参阅4-124 页的“窗口资源”。

当用户创建一个新的文稿或您在处理自己的对话框和警告栏时,使用定位常数是很方便的。但是，当您创建新的窗口来显示一个先前存储的文稿时，如果有可能，新窗口应该在与原有窗口(即上次存储时使用的窗口)同样的矩形中出现。有关窗口放置的规则，请参阅 4-30 页的“在桌面环境中为一个文稿窗口定位”。

请使用GetNewCWindow或GetNewWindow 函数从一个 'WIND' 资源中创建窗口。

凭借资源创建一个窗口.

用户每创建一个新文稿,打开一个先前存储的文稿或执行一个引发对话框的命令时,通常都需要创建一个新窗口。

您可以凭借一个窗口资源通过GetNewCWindow和GetNewWindow 函数来创建文稿窗口。(无论当前安装的是否是彩色显示器，只要使用Color QuickDraw，您就可以使用GetNewCWindow 来创建彩色窗口。一个彩色窗口记录的大小和窗口记录是一样的,并且GetNewCWindow返回一个WindowPtr类型的指针，所以大多数代码能够处理彩色窗口，同样地也能处理单色窗口。)

您可以通过GetNewCWindow为您的窗口记录分配存储器。但是，如果您利用在初始化例程时为控制存储器而指定的内存块自己来分配存储器，然后将指针传递给GetNewCWindow，您可以对存储器的使用保留更多的控制。

通常在您创建文稿窗口时，应根据控制（'CNTL'）资源创建滚动杆，然后在您使窗口可视的同时，显示滚动杆。

程序清单 4-3 说明了一个应用程序定义的过程 DoNewCmd，当用户在文件(File) 菜单中选择“新文稿 (New)”命令时，SurfWriter 调用该过程。在进行创建时，窗口通常是不可视的，只有在所有的元素都就位时才显示出来。

程序清单 4-3 创建一个新窗口

```
PROCEDURE DoNewCmd (newDocument: Boolean; VAR window: WindowPtr);
VAR
    myData:        MyDocRecHnd;        {the document's data record}
    windStorage:   Ptr;                {memory for window record}
    destRect,                          {rectangles for creating }
    viewRect:      Rect;               { TextEdit edit record}
    good:          Boolean;            {success flag}
BEGIN
    window := NIL;                          {no window created yet}
    good := FALSE;                          {no success yet}
    {allocate memory for window record from previously allocated block}
    windStorage := MyPtrAllocationProc;
    IF windStorage <> NIL THEN              {memory allocation succeeded}
    BEGIN                                   {create window}
        IF gColorQDAvailable THEN
            window := GetNewCWindow(rDocWindow, windStorage, WindowPtr(-1))
        ELSE
            window := GetNewWindow(rDocWindow, windStorage, WindowPtr(-1));
    END;
                                            {create document record}
    myData := MyDocRecHnd(NewHandle(SIZEOF(MyDocRec)));
    IF(window <> NIL) AND (myData <> NIL) THEN {window record and document }
    BEGIN                                      { record both allocated}
        SetPort(window);                       {set current port}
        HLock(Handle(myData));                 {lock handle to doc record}
        SetWRefCon(window, LongInt(myData)); {link document record to window}
        WITH window^, myData^^ DO              {fill in document record}
        BEGIN
            MyGetTERect(window, viewRect);     {set up a viewRect for TextEdit}
            destRect := viewRect;
            destRect.right := destRect.left + kMaxDocWidth;
            editRec := TENew(destRect, viewRect);
            IF editRec <> NIL THEN             {it's a good edit record}
            BEGIN
                good := TRUE;                  {set success flag}
                MyAdjustViewRect(editRec);     {set up edit record}
                TEAutoView(TRUE, editRec);
            END
```

```
        ELSE
            good := FALSE;              {clear success flag}
            IF good THEN
            BEGIN                       {create scroll bars}
                vScrollBar := GetNewControl(rVScroll, window);
                hScrollBar := GetNewControl(rHScroll, window);
                good := (vScrollBar <> NIL) AND (hScrollBar <> NIL);
            END;
            IF good THEN                {it's a good document}
            BEGIN
                MyAdjustScrollBars(window, FALSE);  {adjust scroll bars}
                fileRefNum := 0;        {no file yet}
                windowDirty := FALSE;   {no changes yet}
                IF newDocument THEN     {if it's a new (empty) document, }
                    ShowWindow(window);{ make it visible}
            END;
        END;       {end of WITH statement}
        HUnlock(Handle(myData));        {unlock document record}
    END;       {end of IF (window <> NIL) AND (myData <> NIL)}
    IF NOT good THEN
    BEGIN
        IF windStorage <> NIL THEN    {memory for window record was allocated}
            DisposePtr(windStorage);{dispose of it}
        IF myData <> NIL THEN         {memory for document record was allocated}
        BEGIN
            IF myData^^.editRec <> NIL THEN     {edit record was allocated}
                TEDispose(myData^^.editRec);    {dispose of it}
            DisposeHandle(Handle(myData));      {dispose of document record}
        END;
        IF window <> NIL THEN        {window pointer exists, but it's invalid}
            CloseWindow(window);     {clean up window pointer}
        window := NIL;               {set window to NIL to indicate failure}
    END;
END; {DoNewCmd}
```

DoNewCmd过程首先设定窗口指针window为NIL和成功标志good为FALSE来显示一个有效窗口尚不存在。然后它调用应用程序定义的函数MyPtrAllocationProc，该函数为窗口记录分配存储器，此窗口记录位于程序初始化时为存储器分配留出来的一个块中。如果MyPtrAllocationProc成功地分配了存储器，并返回一个有效的指针，则DoNewCmd创建一个窗口，并像常数rDocWindow 所指定的那样，用资源标识符 128 指定‘WIND’资源。使用了这个窗口资源（在 4-26 页的程序清单 4-2 中定义），窗口管理器就能创建一个类型为zoomDocProc的不可视窗口。由于 GetNewCWindow 或 GetNewWindow的behind参数值为 WindowPtr (-1)，所以窗口管理器在桌面环境中将新的窗口放置在所有其它窗口的前面。

接下来，DoNewCmd过程创建一个文稿记录。DoNewCmd在操纵文稿记录时将其锁定在存储器中，将窗口记录中的refCon域设定为指向文稿记录的指针，并填写文稿记录。在填写文稿记录的同时，DoNewCmd设置一个TextEdit 记录来保持用户数据。若成功了，DoNewCmd就设置水平和垂直的滚动杆（关于应用程序定义的过程MyAdjustScrollbars，请参阅本书的“控制管理器”一章），并填写文稿记录的其余部分。若创建的窗口是为了显示一个新文稿，即不需要从磁盘读入用户数据，则DoNewCmd 调用ShowWindow过程，使窗口立即可视。

若您的窗口资源指定一个新的窗口为可视的，则GetNewCWindow 立即显示该窗口。但是，如果您希望创建一个窗口，多数情况下应使创建的窗口处于不可视状态，然后在需要显示它时，令它成为可视窗口。

- 如果您因为用户创建一个新文稿而需要创建窗口，为了使窗口框架可视，您可以调用ShowWindow过程立即显示该窗口。这一可视性的改变添加到更新区域，并触发了一个更新事件。然后您的应用程序请求它自己的过程响应更新事件以绘制内容区域

- 如果您希望创建一个新窗口来显示存储的文稿，那么您必须在显示它之前，检索用户数据。[有关读取存储的文件，请参阅《Inside Macintosh 文件》(*Inside Macintosh: Files*)。] 如果有可能，用来显示文稿的窗口的大小和位置应该和文稿上次存储时一样。(有关窗口放置的讨论，请参阅下一节“在桌面环境中对文稿窗口定位”。) 只要您为窗口进行了定位并设置了它的内容区域，您就可以调用ShowWindow使窗口可视，它可触发一个更新事件。然后您的应用程序请求它自己的过程来绘制内容区域

在桌面环境中对文稿窗口定位

您在桌面环境中对文稿窗口定位的目的是将它放置在用户希望放置的地方。对一个新文稿而言，这一位置通常恰在用户进行工作的上一个文稿窗口的右下面。对于一个存储的文稿，这一位置通常是该文稿上次被存储时的文稿窗口的位置（如果它是在屏幕配置相同的电脑上存储的）。本节介绍文稿窗口的放置。本书的“对话管理器”一章介绍了警告栏和对话框的放置。有关窗口放置的详细说明，请参阅《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*)。

在具有一个已知大小的单个屏幕的 Macintosh 电脑上，对窗口定位是相当简捷的。您可将第一个新文稿窗口定位在桌面环境的左上角。以后每个新窗口的左上角依次在它前面一个窗口左上角的右下方不远处打开。图 4-15 描述了如何在单个屏幕上定位多个文稿。

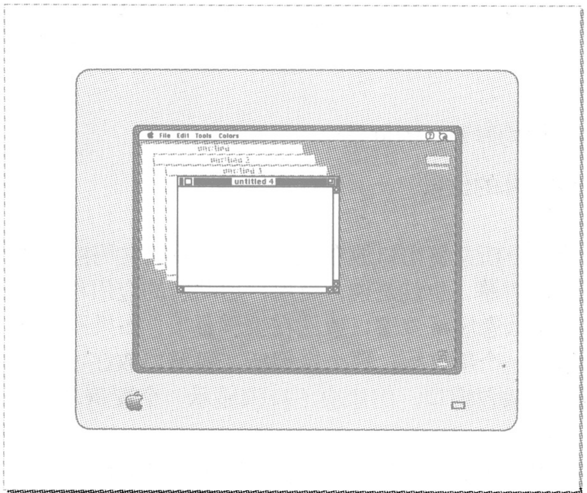


图 4-15　单个屏幕上的文稿窗口位置

若用户关闭了一个或多个文稿窗口，在对右下处添加更多的位置前，应先在“空出的”位置显示后继的窗口。图 4-16 说明了当用户关闭了一个原来创建的窗口后，打开一个新文稿时，您应如何填充空出的位置。

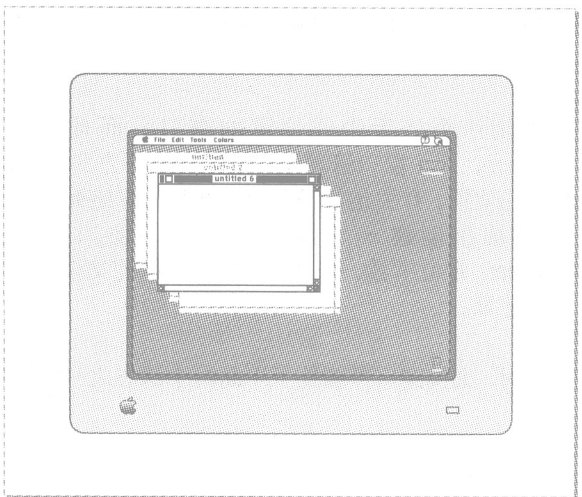


图 4-16　“填充”空出的文稿窗口位置

在带有多个显示器的电脑上，窗口的放置取决于许多因素:

- 可用屏幕的数量和尺寸
- 主屏幕（即包含菜单栏的屏幕）的位置
- 用户最近一次工作的屏幕的位置

一般来说，您把第一个新文稿窗口放在主屏幕，把后继文稿窗口放在包含最新的当前文稿窗口最大部分的屏幕。也就是说，当用户启动您的应用程序时，如果您显示一个空白的文稿窗口，那么您应将该窗口放置在主屏幕上。如果用户将该窗口移动到另一个屏幕，然后创建另一个新文稿，那么您把新文稿窗口放在该屏幕上。虽然用户可以自由地跨过屏幕边界放置窗口，但您决不应该显示一个覆盖多屏幕的新窗口。

当用户打开一个存储的文稿时，如果有可能，您应重复文稿上次存储时窗口的大小和位置。

窗口管理器可识别窗口资源中的一组定位常数，它们使您可以自动对新的窗口定位。通常您可使用常数staggerParentWindowScreen定位文稿窗口。staggerParentWindowScreen常数指定了放置文稿窗口的准则：当从一个包含常数staggerParentWindowScreen的模板创建窗口时，窗口管理器在主屏幕的左上角放置第一个窗口。它把后继窗口的左上角放置在用户上次工作的窗口左上角的右侧 20 像素、下侧20像素的位置。图 4-17 说明了当staggerParentWindowScreen 规范有效，而且用户进行工作的窗口已经不在主屏幕时，窗口管理器如何定位一个新文稿窗口。

如果用户移动或关闭一个处于中间位置的窗口，而且窗口模板指定了staggerParentWindowScreen，那么窗口管理器在继续向右下方移动前，将使用“空”槽作为下一个新创建窗口的位置。有关定位常数及其作用的详细列表，请参阅 4-124 页的“窗口资源”。通常在创建一个显示新文稿的窗口时，您可以使用staggerParentWindowScreen 定位常数。但是，当打开存储的文稿及缩放窗口时，您必须执行自己的窗口位置计算。

当用户存储一个文稿时，文稿窗口可能处于下面两个状态之一：用户状态或标准状态。

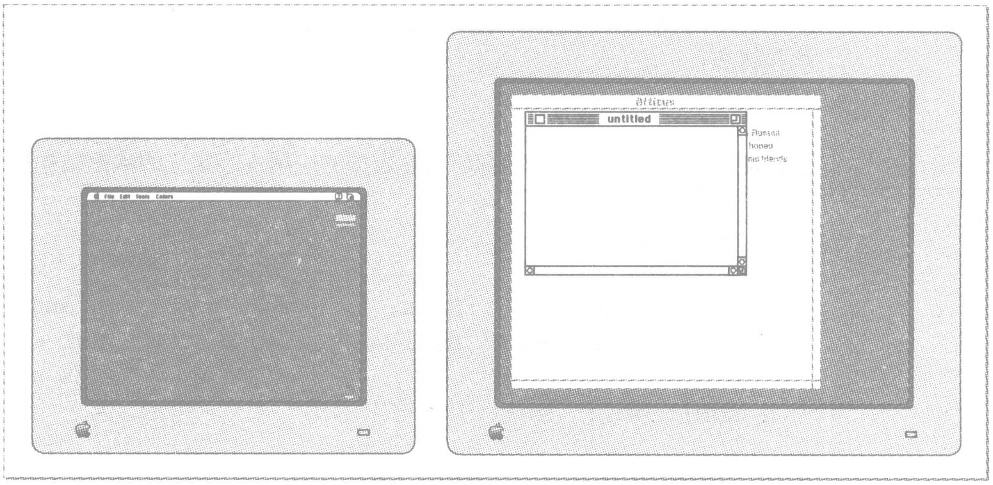


图 4-17　多个屏幕上的文稿窗口位置

用户状态是用户上次为窗口建立的大小和位置。

标准状态是您的应用程序根据文稿的功能和屏幕的可用空间确定的窗口最方便的大小。有关标准状态更详细的说明，请参阅 4-53 页的“缩放一个窗口”内容。通常，每次用户缩放至标准状态时，您的应用程序应对其进行计算。

用户状态和标准状态存储在状态数据记录中，该记录的间接指针出现在窗口记录的 dataHandle 域中。

```
TYPE WStateData =
     RECORD
          userState:     Rect;     {size and location established by user}
          stdState:      Rect;     {size and location established by }
                                   { application}
     END;
```

当用户存储一个文稿时，您必须存储用户状态矩形和窗口的状态(即窗口是处于用户状态还是标准状态)。然后当用户以后再次打开该文稿时，您可以重复窗口的状态。通常您将状态数据作为资源存储在文稿文件的资源分支中。

程序清单 4-4 说明了一个应用程序定义的数据结构，它用于存储窗口的用户矩形和状态。

程序清单 4-4 用于储存窗口状态数据的应用程序定义的数据结构

```
TYPE MyWindowState =
     RECORD
          userStateRect:Rect;      {user state rectangle}
          zoomState:     Boolean; {window state:TRUE = standard; }
                                   {              FALSE = user}
     END;

MyWindowStatePtr   = ^MyWindowState;
MyWindowStateHnd = ^MyWindowStatePtr;
```

当用户存储文稿时，此结构被转换为应用程序定义的资源后存于文稿的资源分支中。

程序清单 4-5 是一个存储文稿状态数据的应用程序例程。当用户存储文稿时，程序将调用该过程 --MySaveWindowPosition。

程序清单 4-5 存储一个文稿窗口的位置

```
PROCEDURE MySaveWindowPosition (myWindow: WindowPtr;
                                 myResFileRefNum: Integer);
VAR
     lastWindowState:         MyWindowState;
     myStateHandle:      MyWindowStateHnd;
     curResRefNum:       Integer;
BEGIN
     {Set user state provisionally and determine whether window is zoomed.}
     lastWindowState.userStateRect := WindowPeek(myWindow)^.contRgn^^.rgnBBox;
     lastWindowState.zoomState := EqualRect(lastWindowState.userStateRect,
                                   MyGetWindowStdState(myWindow));
     {if window is in standard state, then set the window's user state from }
     { the userState field in the state data record}
     IF lastWindowState.zoomState THEN     {window was in standard state}
          lastWindowState.userStateRect := MyGetWindowUserState(myWindow);
     curResRefNum := CurResFile;  {save the refNum of current resource file}
     UseResFile(myResFileRefNum); {set the current resource file}
     myStateHandle := MyWindowStateHnd(Get1Resource(rWinState,
                                             kLastWinStateID));
```

```
    IF myStateHandle <> NIL THEN        {a state data resource already exists}
    BEGIN                               {update it}
        myStateHandle^^ := lastWindowState;
        ChangedResource(Handle(myStateHandle));
    END
    ELSE                                {no state data has yet been saved}
    BEGIN                               {add state data resource}
        myStateHandle := MyWindowStateHnd(NewHandle(SizeOf(MyWindowState)));
        IF myStateHandle <> NIL THEN
        BEGIN
            myStateHandle^^ := lastWindowState;
            AddResource(Handle(myStateHandle), rWinState, kLastWinStateID,
                        'last window state');
        END;
    END;
    IF myStateHandle <> NIL THEN
    BEGIN
        UpdateResFile(myResFileRefNum);
        ReleaseResource(Handle(myStateHandle));
    END;
    UseResFile(curResRefNum);
END;
```

MySaveWindowPosition 过程首先从窗口内容区域的边界矩形设定自己的用户状态字段，并将该矩形与存储在状态数据记录中的用户状态进行比较，以确定窗口是处于用户状态还是标准状态。（若二者匹配，则窗口处于用户状态，否则处于标准状态。）若窗口处于标准状态，则该过程用存储在状态数据记录中userState 域的矩形代替它自己的用户状态数据。该过程的其余部分在文稿的资源分支中存储应用程序定义的状态数据记录。

当创建一个新窗口来显示存储的文稿时，SurfWriter 还原存储的用户状态数据，并重新计算标准状态。但是，在使用存储的矩形之前，SurfWriter 要确认该位置在桌面环境中是可实现的。（例如，若用户在装有多个显示器的电脑上存储一个文稿，然后在只有一个显示器的系统中打开它，那么存储的窗口位置可能全部或部分地偏出了屏幕）。

程序清单 4-6 给出了MySetWindowPosition，这是当用户打开一个存储的文稿时，SurfWriter 调用的应用程序定义的例程。MySetWindowPosition过程检索文稿的存储状态数据，然后调用另一个应用程序定义的例程 MyVerifyPosition，以确认存储的位置是否可实现。

程序清单 4-6 当用户打开一个存储的文稿时对窗口定位

```
PROCEDURE MySetWindowPosition (myWindow: WindowPtr);
VAR
 myData:             MyDocRecHnd;
 lastUserStateRect: Rect;
 stdStateRect:       Rect;
 curStateRect:       Rect;
 myRefNum:           Integer;
 myStateHandle:      MyWindowStateHnd;
 resourceGood:       Boolean;
 savePort:           GrafPtr;
 myErr:              OSErr;
BEGIN
 myData := MyDocRecHnd(GetWRefCon(myWindow)); {get document record}
 HLock(Handle(myData));      {lock the record while manipulating it}
 {open the resource fork and get its file reference number}
 myRefNum := FSpOpenResFile(myData^^.fileFSSpec, fsRdWrPerm);
 myErr := ResError;
 IF myErr <> noErr THEN
     Exit(MySetWindowPosition);
 {get handle to rectangle that describes document's last window position}
 myStateHandle := MyWindowStateHnd(Get1Resource(rWinState,
                                                 kLastWinStateID));
 IF myStateHandle <> NIL THEN        {handle to data succeeded}
 BEGIN    {retrieve the saved user state}
     lastUserStateRect := myStateHandle^^.userStateRect;
     resourceGood := TRUE;
 END
 ELSE
 BEGIN
     lastUserStateRect.top := 0;{force MyVerifyPosition to calculate }
     resourceGood := FALSE;      { the default position}
 END;
 {verify that user state is practical and calculate new standard state}
 MyVerifyPosition(myWindow, lastUserStateRect, stdStateRect);
 IF resourceGood THEN                  {document had state resource}
     IF myStateHandle^^.zoomState THEN{if window was in standard state }
         curStateRect := stdStateRect{ when saved, display it in }
                                       { newly calculated standard state}
     ELSE               {otherwise, current state is the user state}
         curStateRect := lastUserStateRect
 ELSE                                  {document had no state resource}
     curStateRect := lastUserStateRect;{use default user state}
```

```
    {move window}
    MoveWindow(myWindow, curStateRect.left, curStateRect.top, FALSE);
    {Convert to local coordinates and resize window.}
    GetPort(savePort);
    SetPort(myWindow);
    GlobalToLocal(curStateRect.topLeft);
    GlobalToLocal(curStateRect.botRight);
    SizeWindow(myWindow, curStateRect.right, curStateRect.bottom, TRUE);
    IF resourceGood THEN               {reset user state and standard }
    BEGIN                      { state--SizeWindow may have changed them}
        MySetWindowUserState(myWindow, lastUserStateRect);
        MySetWindowStdState(myWindow, stdStateRect);
    END;
    ReleaseResource(Handle(myStateHandle));          {clean up}
    CloseResFile(myRefNum);
    HUnLock(Handle(myData));
END;
```

MyVerifyPosition例程未在这里给出，它将存储的位置与可用的屏幕空间进行比较。(有关将存储的矩形与可用的屏幕空间比较的方法，请参阅 4-55 页的程序清单4-12。) MyVerifyPosition 在必要时改变用户状态矩形（如果有可能，就使用同样的大小，但须把它放在可用的屏幕空间中)，并计算一个新的标准状态。以便在包含用户状态的屏幕上显示窗口。在确定有效的用户状态矩形和标准状态矩形后，过程MySetWindowPosition根据存储文稿时其窗口的状态，可以用适当的大小和位置设定临时定位矩形，然后MySetWindowPosition过程调用窗口管理器过程 MoveWindow 和SizeWindow ，在清除之前建立窗口的位置和大小。

SurfWriter 应用程序在读取文稿数据分支中的数据后，从自己的例程中调用MySetWindowPosition 以便打开存储的文稿。程序清单 4-7 给出了应用程序定义的DoOpenFile函数，当用户打开一个存储的文稿时，SurfWriter调用此函数。

程序清单 4-7 打开一个存储的文稿

```
FUNCTION DoOpenFile (mySpec: FSSpec): OSErr;
VAR
  myWindow:        WindowPtr;
  myData:          MyDocRecHnd;
  myFileRefNum:    Integer;
  myErr:           OSErr;
```

```
BEGIN
    DoNewCmd(FALSE, myWindow); {FALSE tells DoNewCmd not to }
                               { show the window}
    IF myWindow = NIL THEN
    BEGIN
        DoOpenFile := kOpenFileError;
        Exit(DoOpenFile);
    END;
    SetWTitle(myWindow, mySpec.name);
    {open the file's data fork, passing the file spec-- }
    { FSpOpenDF returns a file reference number}
    myErr := FSpOpenDF(mySpec, fsRdWrPerm, myFileRefNum);
    IF (myErr <> noErr) AND (myErr <> opWrErr) THEN{open failed}
    BEGIN                                           {clean up}
        DisposeWindow(myWindow);
        DoOpenFile := myErr;
        Exit(DoOpenFile);
    END;
    {get a handle to the window's document record}
    myData := MyDocRecHnd(GetWRefCon(myWindow));
    myData^^.fileRefNum := myFileRefNum;{save file ref num}
    myData^^.fileFSSpec := mySpec;      {save fsspec}
    myErr := DoReadFile(myWindow);      {read file's data}
    {retrieve saved state data and establish valid position}
    MySetWindowPosition(myWindow);
    {MyResizeWindow invalidates the whole portRgn, guaranteeing }
    { an update event--the window's contents are redrawn then}
    MyResizeWindow(myWindow);
    ShowWindow(myWindow);               {show window}
    DoOpenFile := myErr;
END;
```

DoOpenFile 首先调用应用程序定义的过程 DoNewCmd 来创建一个新窗口，但并不立即显示窗口。(4-28 页的程序清单 4-3 说明了 DoNewCmd 过程。) 然后 DoOpenFile 将窗口标题设定为文稿文件的名称，并读入数据。接着，它调用 MySetWindowPosition 来确定新窗口的位置。在建立一个有效的位置后，DoOpenFile 调用应用程序定义的例程 MyResizeWindow（在 4-59 页的程序清单 4-14 给出）以新的尺寸设置内容区域，最后才使窗口可视。

绘制窗口内容

您的应用程序和窗口管理器可共同完成在屏幕上显示窗口。当您创建一个窗口并使它可视后，窗口管理器就在适当的位置自动绘制窗口框架。当用户对桌面环境进行改变，以便移动窗口及调整不同窗口的大小时，窗口管理器就在必要时改变窗口框架。窗口框架包括窗口轮廓线、标题栏和关闭格及缩放格。

您的应用程序负责绘制窗口内容区域。通常它使用控制管理器来绘制窗口控制部件，使用窗口管理器来绘制尺寸格，并自己绘制用户数据。本章的示例代码使用一个内容区域的简单模型，它只包含控制部件、尺寸格和一个TextEdit 记录。[有关TextEdit的说明，请参阅《Inside Macintosh 文本》(*Inside Macintosh : Text*)。]

程序清单 4-8 说明一个用于绘制窗口内容区域的应用程序定义的过程。

程序清单 4-8　绘制一个窗口

```
PROCEDURE MyDrawWindow (window: WindowPtr);
VAR
    myData: MyDocRecHnd;
BEGIN
    SetPort(window);
    myData := MyDocRecHnd(GetWRefCon(window));
    HLock(Handle(myData));
    WITH window^ DO
    BEGIN
        EraseRect(portRect);   {erase content area}
        UpdateControls(window, visRgn); {draw window controls}
        DrawGrowIcon(window);  {draw size box}
        {update window contents as appropriate to your }
        { application (in this case use TextEdit)}
        TEUpdate(portRect, myData^^.editRec);
    END;
    HUnLock(Handle(myData));
END;
```

MyDrawWindow过程首先将当前端口设定为窗口端口，并获取该窗口文稿记录的间接指针。该过程使用文稿记录中的数据，首先擦去内容区域，再绘制控制部件，绘制尺寸格。最后，它绘制用户数据，在这里是一个 TextEdit 编辑记录的内容。

若您的应用程序创建一个包含静止显示的窗口，您可以对窗口记录中windowPic域中的图片放置一个间接指针，使窗口管理器负责绘制并更新内容区域。请参阅4-110页中对 SetWindowPic 的说明。

更新内容区域

窗口管理器可以保留一个更新区域，以帮助您的应用程序保持窗口的当前显示，该更新区域代表涉及到桌面环境改变的那一部分内容区域。例如，若用户将当前活动窗口拖拽到一个新位置，从而暴露了某个失活窗口的一部分，窗口管理器就把失活窗口新暴露的部分添加到更新区域。

图 4-18 说明当用户暴露了附加的内容区域后，窗口管理器如何把窗口内容区域的一部分添加到它的更新区域。

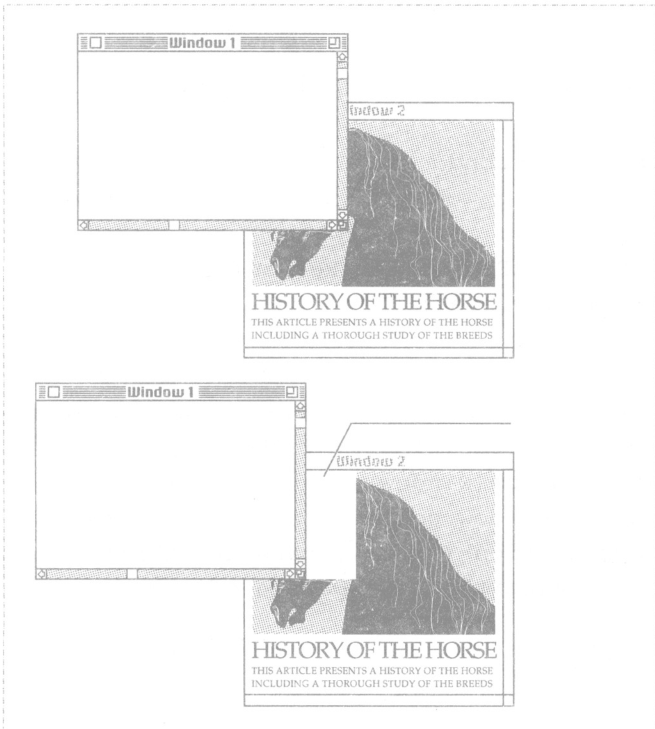


图 4-18　某个窗口的移动造成另一个窗口的更新区域的变动

事件管理器周期性地扫描桌面中所有窗口的更新区域。若它找到一个具有非空更新区域的窗口，它就为那个窗口产生一个更新事件。当您的应用程序接收到一个更新事件时，它对所有需要刷新屏幕的内容区域刷新屏幕，这些在4-48页的“处理更新事件”一节中作了介绍。

当用户改动文稿时，您的应用程序必须同时更新文稿数据和它的窗口内容区域显示的文稿。您可以使用下面两种方法之一来更新显示：

- 若您的应用程序不要求连续的滚动或快速响应，您可以将内容区域改变的部分添加至窗口的更新区域。然后，事件管理器向您的应用程序发送一个更新事件，您的应用程序再请求使用其标准的更新过程
- 对连续滚动和较快的响应时间，您可以直接在窗口内容区域中绘制

在这两种情况下，您的应用程序最终都要在代表窗口的图形端口中进行绘制。您可通过控制管理器绘制控制部件，并使用在《Inside Macintosh 文本》(*Inside Macintosh : Text*) 和《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*) 中介绍的例程绘制文字。

保留更新区域

您的应用程序可以通过操纵更新区域来强制产生及抑制更新事件，这可以使用窗口管理器为此提供的例程来进行。

例如，当用户调整包含尺寸格和滚动杆的窗口大小时，您的应用程序通常操纵更新区域。若用户放大窗口，窗口管理器将新暴露的部分添加到窗口的更新区域，但是不添加原来被滚动杆占据的部分。在调用SizeWindow过程调整窗口大小前，您的应用程序可以两次调用InvalRect过程将滚动杆和尺寸格部分添加到更新区域中。下一次您的应用程序接收到更新事件时，它就擦去滚动杆，并绘制可能在该位置出现的文稿内容的任何部分。

同样地，您可以从更新区域删除一个部分，只要您知道它事实上是有效的。限制更新区域的大小缩短了重新绘制的时间。例如，4-58 页中的程序清单 4-13使用了ValidRect过程从被调整大小的窗口的更新区域中删除未涉及到的文字部分。

处理窗口中的事件

您的应用程序必须准备处理两类有关窗口的事件：

- 在您应用程序窗口中的鼠标和键盘事件，它们是由事件管理器直接响应用户的动作而报告的
- 激活和更新事件，它们是作为用户动作的间接结果，由窗口管理器和事件管理器生成的

在System 7中，如果您的应用程序是前台进程，而且用户单击菜单栏、单击属于您应用程序的窗口或者单击一个在您应用程序的（存储器）分区中启用的桌面附件的窗口，那么您的应用程序都可接收到一个鼠标按下事件。(若用户单击属于另一个应用程序的窗口，事件管理器就向您的应用程序发送一个挂起事件，并执直接那个应用程序的主切换，除非最前面的窗口是一个警告栏或定模式对话框，这时对话管理器仅发出系统示警声，而进程管理器仍将您的应用程序保留为前台进程。)当您的应用程序接收到一个鼠标按下事件时，它首先调用FindWindow函数将光标位置映射到一个窗口区域，然后它转移到自己的一个例程，这些在下节“处理窗口中的鼠标事件”中作了介绍。

当桌面环境或窗口的改变要求窗口内容区域中的一部分或全部都更新时，事件管理器向您的应用程序发送一个更新事件。窗口管理器和您的应用程序都可以通过将需要更新的区域添加到更新区域来触发更新事件，这些在 4-48 页开始的“处理更新事件”一节中作了介绍。

当一个失活窗口变成当前活动窗口或一个当前活动窗口变成失活窗口时，您的应用程序将接收到激活事件。激活事件是您的应用程序和窗口管理器密切合作的实例。当您在应用程序的一个失活窗口中接收到鼠标按下事件时，您可以调用SelectWindow过程，它将取消原当前活动窗口的醒目显示，并为新当前活动窗口添加醒目显示。它还可生成两个激活事件：一个告诉您的应用程序停用原来的当前活动窗口，另一个激活新的现用窗口。然后，您的应用程序就可以激活和释放内容区域，这些在 4-50 页的“处理激活事件”中作了介绍。

当用户第一次单击一个失活窗口时，大多数应用程序在使窗口成为现用之前，不作选定或者改变窗口及文稿。当用户单击您应用程序的一个窗口，使它接收到一个继续执行事件时，您可能甚至不希望接收到使应用程序变成前台进程的按下鼠标事件。您要控制是否应通过 'SIZE' 资源收到这个事件，这些在本书前面的“事件管理器”一章中作了介绍。

处理窗口中的鼠标事件

当您的应用程序跑在前台时，它可接收到发生在菜单栏、它的一个窗口、或任何在其（存储器）分区中启用的桌面附件的窗口中的键盘活动和鼠标按下事件。

当您的应用程序接收到一个鼠标按下事件时，它调用FindWindow函数将光标位置映射到一个窗口区域。

该函数可通过返回下列常数之一而指定区域：

```
CONST    inDesk        = 0;      {none of the following}
         inMenuBar     = 1;      {in menu bar}
         inSysWindow   = 2;      {in desk accessory window}
```

```
inContent     = 3; {anywhere in content region except size}
                   {box if window is active,}
                   {anywhere including size box if window}
                   {is inactive}
inDrag        = 4; {in drag (title bar) region}
inGrow        = 5; {in size box (active window only)}
inGoAway      = 6; {in close box}
inZoomIn      = 7; {in zoom box (window in standard state)}
inZoomOut     = 8; {in zoom box (window in user state)}
```

当光标处在窗口中，同时用户按下鼠标按钮时，FindWindow 不仅返回标识某个窗口区域的常数，而且可设定一个指向该窗口的变量参数。

在 System 7 中，若 FindWindow 返回 inDesk，则说明光标不是在菜单栏或您的一个窗口中，也不在您应用程序的（存储器）分区中启用的桌面附件的窗口中。例如，若光标在窗口框架上，但不在拖拽区域、关闭格或缩放格中，该函数可能返回 inDesk。FindWindow 通常不返回 inDesk 值，您一般可以忽略这个不常见的函数结果。

当光标在菜单栏上（inMenuBar）时若用户按下鼠标按钮，您就可调用自己的例程来显示相应的菜单命令，并允许用户选择菜单命令项目。

只有当光标位于您应用程序的（存储器）分区中启用的桌面附件的窗口时，用户按下鼠标按钮，FindWindow 函数才返回 inSysWindow 值。然后您可以调用 SystemClick 过程，把事件记录和窗口指针传递给它。SystemClick 过程在本书的“事件管理器”一章中作了介绍，它确保了使用适当的桌面附件处理事件。

当光标位于您应用程序的一个窗口中时，如果用户按下鼠标按钮，FindWindow 函数就返回一个其它值。您的响应取决于光标是否位于当前活动窗口，若不在，则取决于当前活动窗口的种类。

当您在当前活动窗口中接收到一个鼠标按下事件时，您可将该事件送到适当的例程，以便改变窗口显示或文稿内容。例如，当光标在缩放格中的同时，用户按下鼠标按钮，您就可调用窗口管理器的函数 TrackBox 来醒目显示缩放格，并跟踪鼠标，直至鼠标按钮被释放。

当您在失活窗口中接收到一个鼠标按下事件时，您的响应取决于当前活动窗口的种类：

- 若当前活动窗口是一个可移动定模式对话框，您应该发出系统示警声，并且不采取其它动作（若当前活动窗口是一个由 ModalDialog 过程处理的定模式对话框，则对话管理器不将该事件传送给您的应用程序，而是自己发出系统警告信号）

- 若当前活动窗口是一个文稿窗口或不定模式对话框，那么您可以调用 SelectWindow，把窗口指针传送给它。SelectWindow 过程可删除原当前活动窗口的醒目显示，将新激活的窗口带到前台，并醒目显示它，然后生成必要的激活事件和更新事件来通知所有被涉及的应用程序，这些应用程序的窗口需要刷新屏幕

程序清单 4-9 给出了一个处理鼠标按下事件的应用程序定义的过程。

程序清单 4-9 处理鼠标按下事件

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
     part:          Integer;
     thisWindow:    WindowPtr;
BEGIN
     part := FindWindow(event.where, thisWindow);    {find out where cursor is}
     CASE part OF
     inMenuBar:               {cursor is in menu bar}
          BEGIN
               {make sure menu items are properly enabled/disabled}
               MyAdjustMenus;
               {let user choose a menu command}
               DoMenuCommand(MenuSelect(event.where));
          END;
     inSysWindow:             {cursor is in a desk accessory window}
          SystemClick(event, thisWindow);
     inContent:               {cursor is in the content region of one }
                              { of your application's windows}
          IF thisWindow <> FrontWindow THEN  {cursor is not in front window}
          BEGIN
               IF MyIsMovableModal(FrontWindow) THEN {front window is }
                    SysBeep(30)                      { movable modal}
               ELSE                      {front window is not movable modal}
                    SelectWindow(thisWindow); {make thisWindow active}
          END
          ELSE                {cursor is in content region of active window}
               DoContentClick(thisWindow, event);{handle event in content region}
     inDrag:                  {cursor is in drag area}
          {if a movable modal is active, ignore click in an inactive title bar}
          IF (thisWindow <> FrontWindow) AND MyIsMovableModal(FrontWindow) THEN
               SysBeep(30)
          ELSE
               {let Window Manager drag window}
               DragWindow(thisWindow, event.where, GetGrayRgn^^.rgnBBox);
     inGrow:                  {cursor is in size box}
          DoGrowWindow(thisWindow, event);      {change window size}
```

```
    inGoAway:              {cursor is in close box}
        {call TrackGoAway to handle mouse until button is released}
        IF TrackGoAway(thisWindow, event.where) THEN
            DoCloseCmd;                   {handle close window}
    inZoomIn, inZoomOut:   {cursor is in zoom box}
        {call TrackBox to handle mouse until button is released}
        IF TrackBox(thisWindow, event.where, part) THEN
            DoZoomWindow(thisWindow, part); {handle zoom window}
    END;        {end of CASE statement}
END;            {end of DoMouseDownEvent}
```

DoMouseDown 过程首先调用FindWindow将光标的位置映射到屏幕的一个部分或窗口一个区域。

若光标在菜单栏上，则 DoMouseDown 调用其它应用程序定义的过程来调整并显示菜单，并接受菜单命令的选择。

若光标是在桌面附件创建的窗口中，则DoMouseDown调用SystemClick过程，它为应用程序中的桌面附件处理鼠标按下事件。

若光标是在一个窗口内容区域，DoMouseDown 首先将 FindWindow 返回的窗口指针和FrontWindow函数返回的结果比较，以核对光标是否在当前的当前活动窗口中。若光标在一个失活窗口中，DoMouseDown核对当前活动窗口是否是一个可移动定模式对话框。(若前台窗口是一个警告栏或位置固定的定模式对话框，则应用程序并不在其它窗口中接收鼠标按下事件。)若当前活动窗口是一个可移动定模式对话框，而且光标在另一个窗口中，DoMouseDown只是发出系统示警声，并等待另一个事件。若当前活动窗口不是一个可移动定模式对话框，则DoMouseDown调用SelectWindow来激活光标所在的窗口。SelectWindow过程对窗口进行必要的重新安排，调整醒目显示，并向应用程序发送一对激活事件以停用原当前活动窗口，激活新的当前活动窗口。DoMouseDown只是激活光标所在的窗口；它不为响应窗口中第一次单击而在新激活的窗口中作出选定。

若光标是在当前活动窗口的内容区域，则 DoMouseDown 过程调用另一个应用程序定义的过程（DoContentClick)，它处理在内容区域中的鼠标事件。

若光标在窗口的拖拽区域，则 DoMouseDown 首先核对当可移动定模式对话框为现用的同时，拖拽区域是否在一个失活窗口中。如果是，则 DoMouseDown 仅发出系统示警声，并等待另一个事件。在任何其它情况下，DoMouseDown 调用窗口管理器过程 DragWindow，以显示窗口的轮廓线，只要用户继续拖拽窗口，它就移动轮廓线，在用户释放鼠标按钮时 DoMouseDown 还调用 MoveWindow，以便在新位置绘制窗口。窗口在它的新位置绘制后，无论以前是否为当前活动窗口，它都变成当前活动窗口。

若光标处于尺寸格中，则DoMouseDown调用另一个应用程序定义的例程（4-58页的程序清单 4-13 中给出的 DoGrowWindow），用来调整窗口的大小。

若在关闭格中按下鼠标，则DoMouseDown调用TrackGoAway函数，该函数可醒目显示关闭格，并跟踪所有鼠标活动，直至用户释放鼠标按钮。只要用户按下鼠标按钮并将光标留在关闭格中，TrackGoAway就可保持关闭格的醒目显示（请参阅图4-19）。若用户将光标移出了关闭格，则TrackGoAway取消其醒目显示。

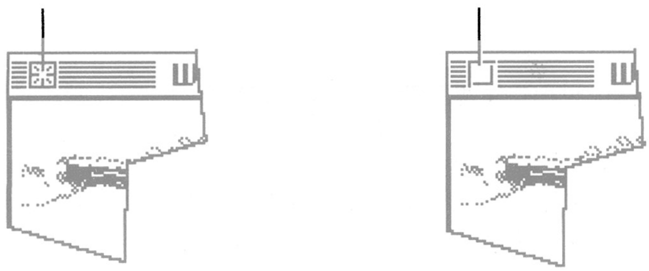


图4-19　醒目显示的关闭格和取消醒目显示的关闭格

当用户释放鼠标按钮时，若光标仍在关闭格中，则 TrackGoAway 返回 TRUE，否则就返回 FALSE。若 TrackGoAway 返回 TRUE，则 DoMouseDown 调用应用程序定义的过程 DoCloseCmd 来关闭窗口。4-60 页的程序清单 4-16 给出了 DoCloseCmd 过程。

若在缩放格中按下鼠标，则DoMouseDown过程首先调用TrackBox，该函数可醒目显示缩放格，并跟踪所有鼠标活动，直至用户释放鼠标按钮。只要用户按住鼠标按钮并把光标留在缩放格中，TrackBox就保留醒目显示的缩放格（请参阅图 4-20）。若用户将光标移出缩放格，则TrackBox取消其醒目显示。

当用户释放鼠标按钮时，若光标仍在缩放格中，则TrackBox返回 TRUE，否则就返回 FALSE。若TrackBox返回 TRUE，则DoMouseDown调用应用程序定义的过程DoZoomWindow 来缩放窗口。4-55 页的程序清单 4-12 给出了DoZoomWindow过程。

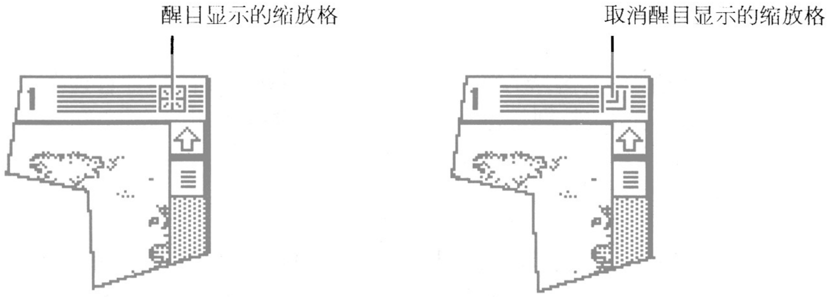


图 4-20　醒目显示的缩放格及取消醒目显示的缩放格

处理窗口中的键盘事件

只要您的应用程序是一个前台进程，它就可接收所有键盘操作事件，但不包括3个标准的“Command”—Shift—数字键序列和用户安装的其它“Command”—Shift—数字键组合。(“Command”—Shift—1 和 “Command”—Shift—2 退出磁盘，“Command”—Shift—3 屏幕照相。您的应用程序不会接收到这些组合键，它们是由事件管理器来处理的。有关的详细介绍，请参阅本书的“事件管理器”一章)。

一般说来，当前活动窗口是键盘操作的目标。

当用户按下一个键或一个组合键时，您应用程序作出的响应是将数据插入文稿、改变显示或采取它定义的其它动作。若要保证对键盘事件使用的一致性并作出响应，就应遵循《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书中规定的准则。例如，您的应用程序应该允许用户按下等效命令组合键（通常是“Command”键和另一个键的组合），以选择经常使用的菜单命令项目。

当您接收到一个键按下事件时，您首先核对用户是否同时按下了一个修正键(标准键盘上的 Command 键、Shift、Control、Caps Lock 和 Option 键）和另一个键。例如，若“Command”键和一个字符键同时被按下，您就调整您的菜单以便适当地启用和停用菜单命令，并允许用户选择与“Command”键组合有关的菜单命令项目。

通常您的应用程序通过在屏幕上绘制字符对标准按键提供反馈。它也应该标识在文字显示中移动光标的箭头键，它还可能添加对功能键或非标准键盘上的其它特殊键的支持。

有关处理键盘事件的应用程序定义的例程的实例，请参阅本书的“事件管理器”一章。

处理更新事件

当您的窗口内容区域的部分或全部需要屏幕刷新时，事件管理器向您的应用程序发送一个更新事件。特别是每当您的应用程序调用 WaitNextEvent 或 EventAvail（或 GetNextEvent）时，事件管理器核对每个窗口的更新区域，并对每个更新区域非空的窗口生成一个更新事件。

通常，当屏幕上窗口的移动和重新安排时均要求对一个或多个窗口刷新屏幕时，窗口管理器触发更新事件。例如，若用户移动一个覆盖了失活窗口一部分的前台窗口时，窗口管理器首先调用失活窗口的窗口定义函数，请求它绘制窗口框架。然后它将新暴露的区域添加到窗口的更新区域，这就触发了一个更新事件，请求您的应用程序来更新内容区域。您的应用程序也可以通过操纵更新区域来触发更新事件。

您的应用程序无论跑在前台还是后台，都可以接收更新事件。

窗口管理器可确保您不会由于将所有屏幕绘图剪贴到窗口图形端口的可视区域，而偶尔发生绘制到另一个窗口中的误动作。可视区域是图形端口的这样一部分，它是在屏幕上确实可以看到的部分—即，没有被其它窗口覆盖的部分。窗口管理器在图形端口数据结构的visRgn域中存储可视区域的间接指针，该数据结构本身是在窗口记录中的。

为了响应一个更新事件，您的应用程序将调用 BeginUpdate 过程，并绘制窗口的内容，然后调用 EndUpdate 过程。如图 4-21 所示，BeginUpdate 将可视区域限制为可视区域和更新区域的重叠部分。然后您的应用程序可以更新整个可视区域或更新整个内容区域—因为QuickDraw限制在可视区域中绘制，只有窗口中确实需要绘制更新的部分。BeginUpdate 过程还可清除更新区域。在您更新窗口后，您可调用 EndUpdate 过程将图形端口的可视区域还原为整个可视区域。

有关图形端口和可视区域的说明，请参阅《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*)。

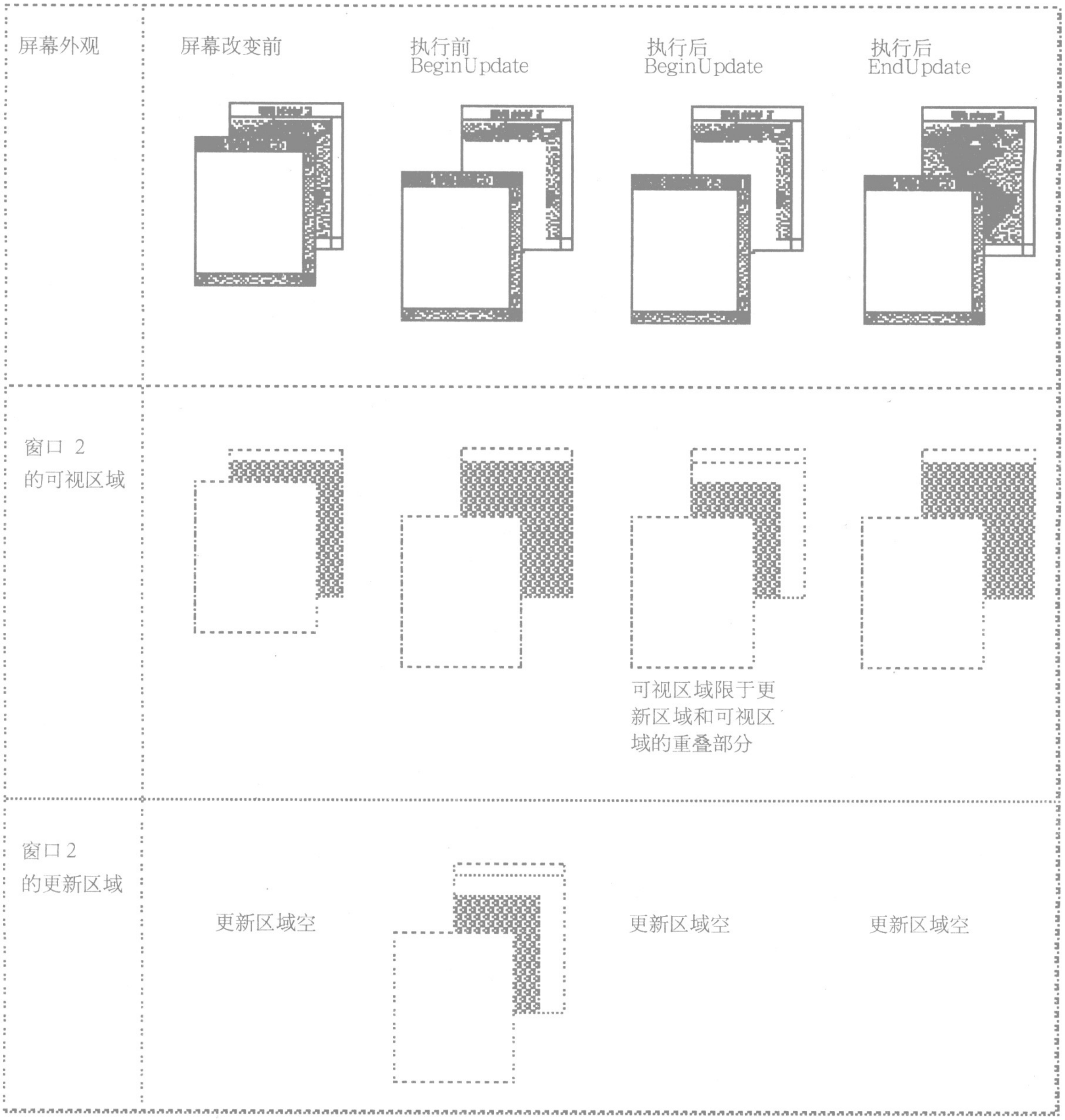


图 4-21　BeginUpdate 和 EndUpdate 对可视区域及更新区域的影响

程序清单 4-10 给出了一个应用程序定义的过程 DoUpdate，它用来处理一个更新事件。

程序清单 4-10 处理更新事件

```
PROCEDURE DoUpdate (window: WindowPtr);
VAR
     windowType: LongInt;
BEGIN
     {determine type of window as defined by this application}
     windowType := MyGetWindowType(window);
     CASE windowType OF
          kMyDocWindow:          {document window}
               BEGIN
                    BeginUpdate(window);
                    MyDrawWindow(window);
                    EndUpdate(window);
               END;
          OTHERWISE              {alert or dialog box}
               DoUpdateMyDialog(window);
               END; {of CASE}
END;
```

DoUpdate 过程首先调用应用程序定义的过程 MyGetWindowType（在 4-25 页的程序清单4-8 中给出）来确定被更新的窗口是一个文稿窗口还是其它某个应用程序定义的窗口。若该窗口是一个文稿窗口，则 DoUpdate 调用 BeginUpdate 创建一个临时可视区域，并调用应用程序定义的过程 MyDrawWindow（在4-39页的程序清单4-8 中给出）来对内容区域刷新屏幕，然后调用 EndUpdate 过程来还原可视区域。

若窗口是一个警告栏或对话框，则 DoUpdate 调用应用程序定义的过程 DoUpdateMyDialog，该过程未在这里给出。

处理激活事件

您的应用程序响应激活事件来激活或停用窗口，激活事件是由窗口管理器生成的，它通知您的应用程序某个窗口将成为当前或失活窗口。每个激活事件指定了要改变的窗口以及改变的方向（即它是否将被激活或被停用）。

您的应用程序常常通过调用 SelectWindow 过程自己触发激活事件。例如，当它在失活窗口中接收到一个鼠标按下事件时，您的应用程序调用 SelectWindow，该过程可将选定的窗口带到前面，取消原窗口的醒目显示，并给选定的窗口添加醒目显示。然后，SelectWindow 过程生成两个激活事件：第一个通知您的应用程序停用原来的当前活动窗口；第二个激活新的当前活动窗口。

当您接收到关于原当前活动窗口的事件后，您应该完成以下各项任务:

- 隐藏控制部件和尺寸格
- 取消或改变该窗口选定项的醒目显示

当您接收到关于新的当前活动窗口的事件后，您应该完成以下各项任务:

- 绘制控制部件和尺寸格
- 还原必要的内容区域，将插入点放在它以前的位置或醒目显示原来突出显示的选定项

若新被激活的窗口也需要更新时，您的应用程序还将接收到一个更新事件，这些已在上一节“处理更新事件”中介绍过。

附　注:
从另一个应用程序到您应用程序窗口的交换是通过挂起事件和恢复执行事件来处理的，而不是通过激活事件。有关您的应用程序如何共享 CPU 时间的说明，请参阅本书的“事件管理器”一章。

程序清单 4-11 给出了应用程序定义的过程 DoActivate，它用来处理激活事件。

程序清单 4-11 处理激活事件

```
PROCEDURE DoActivate (window: WindowPtr; activate: Boolean;
                      event: EventRecord);
VAR
    windowType:          Integer;
    myData:              MyDocRecHnd;
    growRect:            Rect;
BEGIN
    {determine type of window as defined by this application}
    windowType := MyGetWindowType(window);
    CASE windowType OF
        kMyFindModelessDialogBox:       {modeless Find dialog box}
            DoActivateFindDBox(window, event);
                          {modeless Check Spelling dialog box}
        kMyCheckSpellingModelessDialogBox:
            DoActivateCheckSpellDBox(window, event);
        kMyDocWindow:                {document window}
            BEGIN
              myData := MyDocRecHnd(GetWRefCon(window)); {get document record}
              HLock(Handle(myData));         {lock document record}
              WITH myData^^ DO
              IF activate THEN               {window is becoming active}
```

```
        BEGIN
            {restore selections and insert caret--if using }
            { TextEdit, for example, call TEActivate}
            TEActivate(editRec);
            MyAdjustMenus;          {adjust menus for window}
                                    {handle the controls}
            docVScroll^^.contrlVis := kControlVisible;
            docHScroll^^.contrlVis := kControlVisible;
            InvalRect(docVScroll^^.contrlRect);
            InvalRect(docHScroll^^.contrlRect);
            growRect := window^.portRect;
            WITH growRect DO  {handle the size box}
                BEGIN         {adjust for the scroll bars}
                    top := bottom - kScrollbarAdjust;
                    left := right - kScrollbarAdjust;
                END;
            InvalRect(growRect);
        END
        ELSE            {window is becoming inactive}
        BEGIN
            TEDeactivate(editRec);        {call TextEdit to deactivate data}
            HideControl(docVScroll);      {hide the scroll bars}
            HideControl(docHScroll);
            DrawGrowIcon(window);         {draw the size box}
        END;
      HUnLock(Handle(myData));            {unlock document record}
    END; {of kMyDocWindow statement}
  END; {of CASE statement}
END;
```

DoActivate过程首先确定窗口的一般类型，即它调用一个应用程序定义的函数来返回一个标识窗口类型的常数，它可能是查找对话框、核对拼写对话框，或文稿窗口。4-25 页的程序清单 4-1 给出了 MyGetWindowType 函数。

若激活事件的目标是一个对话框窗口，则DoActivate调用其它应用程序定义的例程来激活或停用这些对话框。DoActivateFindDBox 和 DoActivateCheckSpellBox 例程未在这里给出。(DoActivate过程不查找警告栏和定模式对话框，因为对话管理器的 ModalDialog 过程会自动处理这类激活事件。)

若其目标是一个文稿窗口，而且激活事件指定该窗口成为当前活动窗口，DoActivate就突出显示用户在该窗口中的所有选定项，并适当地绘制插入点。然后它使控制部件可视，将滚动杆和尺寸格占据的空间添加到更新区域中。(在更新区域中放置窗口区域保证了一个更新事件。当应用程序收到更新事件时，它调用应用程序定义的DoActivate过程来绘制更新区域，在这里更新区域包括了尺寸格和滚动杆。)

若其目标是一个文稿窗口，而且激活事件指定该窗口成为失活窗口，DoActivate过程就调用TextEdit的过程TEDeactivate来删除用户选定项的醒目显示，调用控制管理器过程HideControl 来隐藏滚动杆，并调用窗口管理器过程DrawGrowIcon来绘制尺寸格和滚动杆区域的轮廓线。

移动窗口

当用户通过按住标题栏(除了关闭格和缩放格区域外)中的光标拖拽一个窗口时，窗口应该跟随光标在桌面环境中移动。您的应用程序可以调用DragWindow过程，使用户能够方便地移动窗口。

DragWindow过程在屏幕上绘制窗口的轮廓线，并在用户移动鼠标时移动轮廓线。当用户释放鼠标按钮时，DragWindow就调用MoveWindow函数，以便在新的位置重新绘制窗口。

有关移动窗口的实例，请参阅 4-44页的程序清单 4-9 中inDrag的介绍。

缩放窗口

缩放格允许用户迅速地在两个窗口的位置和大小状态间交替 用户状态和标准状态。

用户状态是用户建立的窗口大小和位置。若您的应用程序不支持初始用户状态，那么用户状态就仅是用户创建窗口时的大小和位置，直至用户调整该窗口的大小。

标准状态是您的应用程序在考虑文稿的功能和可用的屏幕空间后认为最方便的大小和位置。例如，在一个文字处理应用程序中，标准状态窗口在可能的情况下可以显示一个完整页面，或者显示全宽度、屏幕能容纳的最长长度的一个页面。若用户通过页面设置改变页面的大小，应用程序就可以调整标准状态来反映新的页面大小。若您的应用程序不定义标准状态，窗口管理器就自动将标准状态设定为主屏幕的整个灰度区域，每边减去3像素的边框 [有关您的应用程序如何确定打开和缩放窗口的位置的详细说明，请参阅《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 一书]。用户不能改变窗口的标准状态。

用户状态和标准状态存储在一个记录中，该记录的间接指针位于窗口记录的dataHandle域中。

```
TYPE WStateData =
     RECORD
          userState:Rect;      {size and location established by user}
          stdState: Rect;      {size and location established by }
                               { application}
     END;
```

窗口管理器在填写窗口记录时设定userState和stdState域的初始值，只要用户调整窗口大小，窗口管理器就更新userState域。通常用户每次缩放到标准状态时您要计算标准状态，以保证您缩放到一个正确的位置。

当光标在缩放格中时，如果用户按下了鼠标按钮，FindWindow函数就指定窗口是在用户状态还是在标准状态：当窗口在标准状态时，FindWindow返回 inZoomIn（表示窗口将被缩放“入”用户状态）；当窗口在用户状态时，FindWindow返回inZoomOut（表示窗口将被缩放“出”标准状态）。

当FindWindow返回inZoomIn和inZoomOut时，您的应用程序可以调用TrackBox函数来处理缩放格的醒目显示，并确定当按钮释放时光标是否在缩放格内。若TrackBox返回TRUE，您的应用程序可以调用ZoomWindow过程来调整窗口的大小（在计算了一个新的标准状态后）。若TrackBox返回 FALSE，您的应用程序不需要作任何事情。4-44 页的列表 4-9说明了在事件处理例程中如何使用TrackBox。

程序清单 4-12 给出了一个应用程序定义的过程 DoZoomWindow，应用程序可以在FindWindow返回inZoomIn或inZoomOut后且TrackBox返回TRUE时调用它。因为在窗上次被缩放以后，用户可能已经将该窗口移动到了一个不同的屏幕，所以此过程首先决定那一个屏幕包含了窗口最大的面积，然后在缩放窗口前，对该屏幕计算理想的窗口大小。

DoZoomWindow过程中的屏幕计算取决于处理图形设备的例程，它们是和Color QuickDraw一起引入的。因此，DoZoomWindow 在把要缩放的窗口和设备列表中的图形设备进行比较之前，先核对Color QuickDraw是否存在。若不能用 Color QuickDraw，则DoZoomWindow就假定它是在一个只有单个屏幕的电脑上运行。

程序清单 4-12　　缩放窗口

```
PROCEDURE DoZoomWindow (thisWindow: windowPtr; zoomInOrOut: Integer);
VAR
    gdNthDevice, gdZoomOnThisDevice:    GDHandle;
    savePort:                           GrafPtr;
    windRect, zoomRect, theSect:        Rect;
    sectArea, greatestArea:             LongInt;
    wTitleHeight:                       Integer;
    sectFlag:                           Boolean;
BEGIN
    GetPort(savePort);
    SetPort(thisWindow);
    EraseRect(thisWindow^.portRect);    {erase to avoid flicker}
    IF zoomInOrOut = inZoomOut THEN     {zooming to standard state}
    BEGIN
        IF NOT gColorQDAvailable THEN   {assume a single screen and }
        BEGIN                           { set standard state to full screen}
            zoomRect := screenBits.bounds;
            InsetRect(zoomRect, 4, 4);

WStateDataHandle(WindowPeek(thisWindow)^.dataHandle)^^.stdState
                                                := zoomRect;
        END
        ELSE                {locate window on available graphics devices}
        BEGIN
            windRect := thisWindow^.portRect;
            LocalToGlobal(windRect.topLeft);
                    {convert to global coordinates}
            LocalToGlobal(windRect.botRight);
            {calculate height of window's title bar}
            wTitleHeight := windRect.top - 1 -
                        WindowPeek(thisWindow)^.strucRgn^^.rgnBBox.top;
            windRect.top := windRect.top - wTitleHeight;
            gdNthDevice := GetDeviceList;
            greatestArea := 0;              {initialize to 0}
            {check window against all gdRects in gDevice list and remember }
            { which gdRect contains largest area of window}
            WHILE gdNthDevice <> NIL DO
            IF TestDeviceAttribute(gdNthDevice, screenDevice) THEN
                IF TestDeviceAttribute(gdNthDevice, screenActive) THEN
                BEGIN
                    {The SectRect routine calculates the intersection }
                    { of the window rectangle and this gDevice }
                    { rectangle and returns TRUE if the rectangles intersect, }
                    { FALSE if they don't.}
```

```
            sectFlag := SectRect(windRect, gdNthDevice^^.gdRect,
                                 theSect);
            {determine which screen holds greatest window area}
            {first, calculate area of rectangle on current device}
            WITH theSect DO
                 sectArea := LongInt(right - left) * (bottom - top);
            IF sectArea > greatestArea THEN
            BEGIN
                 greatestArea := sectArea;    {set greatest area so far}
                 gdZoomOnThisDevice := gdNthDevice;{set zoom device}
            END;
            gdNthDevice := GetNextDevice(gdNthDevice);
       END;      {of WHILE}
    {if gdZoomOnThisDevice is on main device, allow for menu bar height}
    IF gdZoomOnThisDevice = GetMainDevice THEN
         wTitleHeight := wTitleHeight + GetMBarHeight;
    WITH gdZoomOnThisDevice^^.gdRect DO  {create the zoom rectangle}
    BEGIN
         {set the zoom rectangle to the full screen, minus window title }
         { height (and menu bar height if necessary), inset by 3 pixels}
         SetRect(zoomRect, left + 3, top + wTitleHeight + 3,
                        right - 3, bottom - 3);
         {If your application has a different "most useful" standard }
         { state, then size the zoom window accordingly.}
         {set up the WStateData record for this window}
         WStateDataHandle(WindowPeek(thisWindow)^.dataHandle)^^.stdState
                                                       := zoomRect;
         END;
    END;
 END; {of inZoomOut}
 {if zoomInOrOut = inZoomIn, just let ZoomWindow zoom to user state}
 {zoom the window frame}
 ZoomWindow(thisWindow, zoomInOrOut, (thisWindow = FrontWindow));
 MyResizeWindow(thisWindow); {application-defined window-sizing routine}
 SetPort(savePort);
END; (of DoZoomWindow)
```

如果用户将窗口缩放到标准状态，DoZoomWindow 根据应用程序本身的情况，当前窗口的位置以及可用的屏幕，来计算新的标准大小和位置。DoZoomWindow 过程总是在显示当前窗口的屏幕上放置标准状态，如果该窗口覆盖屏幕，则在包含最大面积窗口的屏幕上放置标准状态。

程序清单4-12中的代码段用于确定哪个屏幕应显示窗口的标准状态。在此所示的示例代码可以建立标准的状态，它仅占据被选择屏幕的灰度面积，所有的边减去三个像素。您的应用程序应建立适用其自己文稿的标准状态。当计算标准状态时，要尽可能少地从用户状态 移动窗口。如果可能的话，应将标准状态矩形的一个角定位到用户状态矩形的一个角上。

如果用户将窗口缩放到用户状态 ，DoZoomWindow 就不需要任何计算，因为在状态数据记录中存储的用户状态矩形应代表一个有效的屏幕位置。

在计算标准状态后，如果需要，DoZoomWindow 调用 ZoomWindow 过程，用新的大小和位置对窗口框架进行屏幕刷新，然后调用应用程序定义的过程MyResize Window来对窗口的内容区域进行屏幕刷新．第 4 - 5 9 页上的程序清单 4 - 1 4 介绍了MyResizeWindow 过程。

调整窗口大小

位于窗口内容区域右下角中的尺寸格允许用户更改窗口的大小。

当用户在尺寸格中定位光标并按下鼠标按钮时，您的应用程序可以调用窗口管理器的GrowWindow 函数。 此函数可以显示一个增长图像——窗口框架的灰色轮廓线和滚动杆区域， 并可随着用户拖拽尺寸格而扩大或缩小。增长图像表示用户在任意时刻释放鼠标按钮时，该窗口边缘的位置。

若要避免无法控制大小窗口， 那么当您调用GrowWindow时，可以提供大小的上限和下限。用于 GrowWindow 的 sizeRect 参数在 Rect 类型的单结构中指定大小的上限和下限。在 sizeRect 结构中的值代表窗口的尺寸， 而不是屏幕坐标:

- 您在 sizeRect. top 中提供最小垂直度量
- 您在 sizeRect. left 中提供最小水平度量
- 您在 sizeRect. bottom 中提供最大垂直度量
- 您在 sizeRect. right 中提供最大水平度量

多数应用程序指定的最小尺寸都足以将结构区域的所有部分和滚动杆包含在其中。因为用户不能在屏幕边缘以外移动光标，所以您可以放心地将最大尺寸设定成最大的矩形。

当用户释放鼠标按钮时，GrowWindow 返回包含窗口的新高度（在高位字中）和宽度（在低位字中）的一个长整数。数值 0 表示该窗口的大小不变。当 GrowWindow 返回0以外的任何值时， 您就调用 SizeWindow 调整窗口的大小。

附 注

使用实用程序函数 HiWord 和 LoWord 可以分别检索高位字和低位字。

当您更改窗口的大小时，您必须抹除并刷新该窗口的滚动杆。

程序清单4-13举例介绍了用于追踪尺寸格中鼠标活动并调整窗口大小的应用程序定义的过程 DoGrowWindow。

程序清单 4-13 调整窗口大小

```
PROCEDURE DoGrowWindow (thisWindow: windowPtr;
                         event: EventRecord);
VAR
    growSize:     LongInt;
    limitRect:    Rect;
    oldViewRect:  Rect;
    locUpdateRgn: RgnHandle;
    theResult:    Boolean;
    myData:       MyDocRecHnd;
BEGIN
    {set up the limiting rectangle: kMinDocSize = 64 }
                                  { kMaxDocSize = 65535}
    SetRect(limitRect, kMinDocSize, kMinDocSize, kMaxDocSize,
            kMaxDocSize);
    {call Window Manager to let user drag size box}
    growSize := GrowWindow(thisWindow, event.where, limitRect);
    IF growSize <> 0 THEN {if user changed size, }
    BEGIN                 { then resize window}
        myData := MyDocRecHnd(GetWRefCon(thisWindow));
        oldViewRect := myData^^.editRec^^.viewRect;
        locUpdateRgn := NewRgn;
        {save update region in local coordinates}
        MyGetLocalUpdateRgn(thisWindow, locUpdateRgn);
        {resize the window}
        SizeWindow(thisWindow, LoWord(growSize), HiWord(growSize),
                        TRUE);
        MyResizeWindow(thisWindow);
        {find intersection of old viewRect and new viewRect}
        theResult := SectRect(oldViewRect,
                        myData^^.editRec^^.viewRect,
                        oldViewRect);
        {validate the intersection (don't update)}
        ValidRect(oldViewRect);
```

```
        {invalidate any prior update region}
        InvalRgn(locUpdateRgn);
        DisposeRgn(locUpdateRgn);
    END;
END;
```

当光标位于尺寸格时如果按下鼠标按钮，那么处理鼠标按下事件的过程（DoMouseDown，如第4-44页所示）可调用应用程序定义的DoGrowWindow过程。DoGrowWindow过程调用窗口管理器的函数GrowWindow，只要按钮被按住，该函数就可跟踪鼠标的活动。如果用户在释放鼠标按钮之前拖拽尺寸格，GrowWindow就返回一个非零的值，并且GrowWindow准备调整窗口的大小，首先，GrowWindow在变量oldViewRect中存储当前显示的矩形。当屏幕以新的尺寸刷新窗口的内容区域时将用到此信息。GrowWindow过程还把在局部坐标中的当前更新区域存储在LocUpdateRgn区域中，因此在进行其本身的更新区域维护后可以还原该更新区域。(仅当应用程序允许用户输入累加至更新区域进行绘制以响应更新事件，而不是立即绘制到窗口中时,才需要此步骤。)

在存储了当前显示矩形和当前更新区域后,DoGrowWindow就能调用窗口管理器的过程SizeWindow以便用其新的尺寸绘制窗口。然后DoGrowWindow过程调用应用程序定义的过程MyResizeWindow,将窗口的滚动杆和窗口内容调整为新的大小。程序清单4-14举例说明了应用程序定义的MyResizeWindow过程。

在调用SizeWindow后，DoGrowWindow计算旧显示矩形和新显示矩形的交集。它使用此区域使窗口中未更改部分重新有效（即将整个窗口置于更新区域)。这样，当应用程序接收到它的下一个更新事件时，只需重新绘制内容区域的更改部分。

程序清单 4-14　当调整窗口大小时调整滚动杆和内容区域

```
PROCEDURE MyResizeWindow (window: WindowPtr);
BEGIN
    WITH window^ DO
    BEGIN
        {adjust scroll bars and contents-- }
        { see the chapter "ontrol Manager" for implementation}
        MyAdjustScrollbars(window, TRUE);
        MyAdjustTE(window);
        {invalidate content region, forcing an update}
        InvalRect(portRect);
    END;
END; {MyResizeWindow}
```

程序清单 4-15 举例说明了应用程序定义的过程 MyGetLocalUpdateRgn，它可以在当前窗口的局部坐标中提供窗口的更新区域。MyGetLocalUpdateRgn 过程使用 QuickDraw 的例程 CopyRgn 和 OffsetRgn，请参阅《Inside Macintosh 绘图》(*Inside Macintosh: Imaging*)。

程序清单 4-15 将窗口区域转换为局部坐标

```
PROCEDURE MyGetLocalUpdateRgn (window: WindowPtr;
                               localRgn: RgnHandle);
BEGIN
    {save old update region}
    CopyRgn(WindowPeek(window)^.updateRgn, localRgn);
    WITH window^.portBits.bounds DO
        OffsetRgn(localRgn, left, top);    {convert to local coords}
END; {MyGetLocalUpdateRgn}
```

关闭窗口

用户关闭窗口的方法是在左上角单击关闭格，或从 File 菜单中选择 Close 命令项。

当光标位于关闭格上时，如果用户按下鼠标按钮，那么您的应用程序将调用 TrackGoAway 函数来追踪鼠标，直到用户释放按钮，如第 4-44 页程序清单 4-9 所示。当光标位于关闭格之外时，如果用户释放按钮，那么 TrackGoAway 将返回 FALSE，而且您的应用程序不予响应。如果 TrackGoAway 返回 TRUE，那么您的应用程序将请求其自己的过程来关闭窗口。

窗口中所含信息的类型及其内容是否需要存储将决定您关闭一个窗口的特定步骤。本章中的示例代码可识别四种窗口：包含查找对话的不定模式对话框、包含拼写检查 (Spell Check) 对话的不定模式对话框、标准文稿窗口、以及在应用程序分区打开的桌面附件窗口。

程序清单 4-16 举例说明了由一个应用程序定义的过程 DoCloseCmd。该过程可确定关闭何种窗口，并遵循相应的策略。当用户单击一个窗口的关闭格或从文件菜单选择 Close 时，应用程序将调用 DoCloseCmd。

程序清单 4-16 处理关闭命令

```
PROCEDURE DoCloseCmd;
VAR
    myWindow:     WindowPtr;     {pointer to window's record}
    myData:       MyDocRecHnd;   {handle to a document record}
    windowType:   Integer;       {application-defined window type}
```

```
    BEGIN
    myWindow := FrontWindow;
    windowType := MyGetWindowType(myWindow);
    CASE windowType OF
        kMyFindModelessDialog:     {for modeless dialog boxes, }
            HideWindow(myWindow);  { hide window}
        kMySpellModelessDialog:    {for modeless dialog boxes, }
            HideWindow(myWindow);  { hide window}
        kDAWindow:          {for desk accessories, close the DA}
            CloseDeskAcc(WindowPeek(myWindow)^.windowKind);
        kMyDocWindow:       {for documents, handle file first}
            BEGIN
                myData := MyDocRecHnd(GetWRefCon(myWindow));
                MyCloseDocument(myData);
            END;
    END;            {of CASE}
END;
```

DoCloseCmd过程首先确定哪个窗口是当前活动窗口,然后调用由应用程序定义的函数MyGet WindowType,以便按应用程序的定义标识窗口的类型。如果该窗口是不定模式对话框, MyGetWindow仅仅隐藏窗口,使得数据结构留在存储器中。有关显示隐藏窗口的示例例程,请参阅第4 - 64页上的程序清单4-18 。

如果窗口与桌面附件相关,DoCloseCmd 过程调用CloseDeskAcc过程来关闭桌面附件。此情况仅针对兼容性而言;在 System 7中, 桌面附件很少在应用程序分区中打开。

D oCloseCmd读该文稿记录, 然后调用应用程序定义的过程 CloseDocument 来处理关闭文稿窗口。程序清单4-17是MyCloseDocument过程的范例 。

程序清单 4-17　关闭一个文稿

```
PROCEDURE MyCloseDocument (myData: MyDocRecHnd);
VAR
    title:      Str255;       {window/document title}
    item:       Integer;      {item in Save Alert dialog box}
    docWindow:  WindowPtr;    {pointer to window record}
    event:      EventRecord;  {dummy record for DoActivate}
    myErr:      OSErr;        {variable for error-checking}
BEGIN
    docWindow := FrontWindow;
    IF (myData^^.windowDirty) THEN {changed since last save}
```

```
    BEGIN
        GetWTitle(docWindow, title);     {get window title}
        ParamText(title, '', '', '');   {set up dialog text}
        {deactivate window before displaying Save dialog}
        DoActivate(docWindow, FALSE, event);
        {put up Save dialog and retrieve user response}
        item := CautionAlert(kSaveAlertID, @MyEventFilter);
        IF item = kCancel THEN      {user clicked Cancel}
            Exit(MyCloseDocument); {exit without closing}
        IF item = kSave THEN        {user clicked Save}
            DoSaveCmd;              {save the document}
        {otherwise user clicked Don't Save-- }
        { close document in either case}
        myErr := DoCloseFile(myData); {close document}
        {Add your own error handling.}
    END;
    {close window whether or not user saved}
    CloseWindow(docWindow);         {close window}
    DisposePtr(Ptr(docWindow));     {dispose of window record}
END;
```

MyCloseDocument 过程可核查文稿记录中的 windowDirty 域 （在 4-32 页“管理多窗口”中介绍 ）。如果 windowDirty 的值是 TRUE， MyCloseDocument 显示一个对话框，使用户有机会在关闭窗口之前存储该文稿。该对话框使用户在关闭窗口之前可以选择取消此次关闭动作，关闭窗口前存储文稿或不做存储地退出编辑环境。如果用户选择取消项，MyCloseDocument 仅仅退出运行即可。 如果用户选择存储文稿， 则 MyCloseDocument 调用由应用程序定义的例程 DoSaveCmd，此处未给出其范例。[有关如何存储和关闭文件的说明，请参阅《Inside Macintosh 文件》(*Inside Macintosh: Files*) 的介绍文件管理器一章。] 不论用户在关闭窗口之前是否存储文稿，MyClose-Document最终都关闭该文稿，最后从屏幕中消去该窗口并删除分配给窗口记录的存储器空间。

隐藏和显示窗口

每当用户单击窗口的关闭格时，您就可从屏幕上消去该窗口。但有时，您会发现仅仅隐藏窗口而不取消其数据结构的效率更高。

例如，如果您的应用程序中包含用于搜索字符串的查找不定模式对话框，您可能希望在用户工作期间将结构保留在存储器中。当用户单击关闭格来关闭对话框时，您只需调用 HideWindow 过程来隐藏该窗口。当用户下一次选择查找命令时，您的对话框窗口已出现于原位置，而且其查找内容与最后一次使用时所选定的一致。

若要逆向使用 HideWindow 过程，您必须调用 ShowWindow 使窗口可视，并调用 SelectWindow，使其成为当前活动窗口。图4 - 22是此三个过程如何影响屏幕上窗口状态的图示。

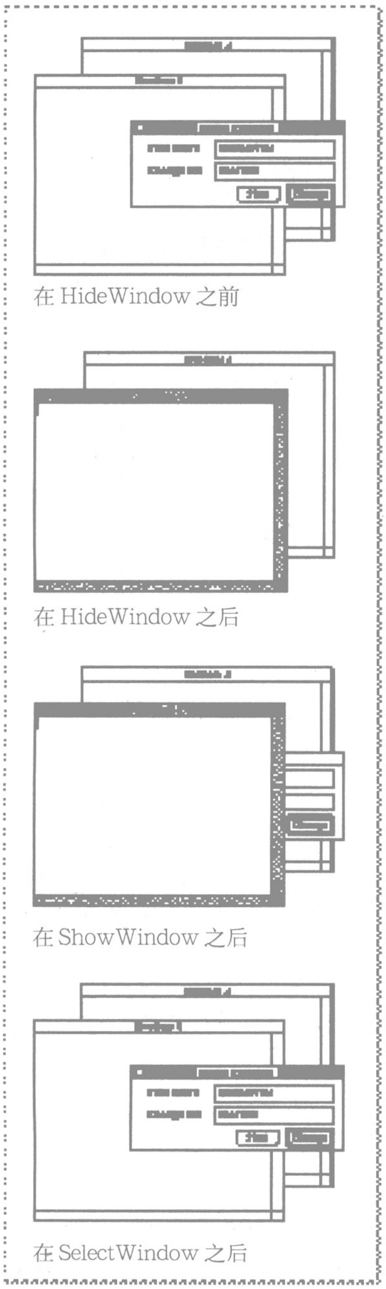


图4-22　HideWindow、 ShowWindow 和 SelectWindow 对窗口的累计影响

在第4－60页中介绍了用于关闭窗口的由应用程序定义的过程DoCloseCmd，它能够在用户关闭窗口时将查找和拼写检查（Spell Check）对话框隐藏起来。程序清单4－18举例说明了一个由应用程序定义的过程 DoShowModelessFindDialogBox，当用户下次选择查找命令时，它可用于重新显示查找对话框。

程序清单4-18 显示隐藏的对话框

```
PROCEDURE DoShowModelessFindDialogBox;
BEGIN
    IF gFindDialog = NIL THEN  {no Find dialog box exists yet}
    BEGIN
        {create Find dialog box}
        gFindDialog := GetNewDialog(rFindModelessDialog, NIL,
                                    Pointer(-1));
        IF gFindDialog = NIL THEN       {creation failed}
            Exit(DoShowModelessFindDialogBox);  {exit}
        {store value that identifies dbox in window refCon field}
        SetWRefCon(gFindDialog, LongInt(kMyFindModelessDialog))
        ShowWindow(gFindDialog);    {make dialog box visible}
    END
    ELSE                {dialog box already exists}
    BEGIN
        ShowWindow(gFindDialog);              {make it visible}
        SelectWindow(gFindDialog);            {select it}
    END;
END;
```

DoShowModelessFindDialogBox过程首先检查查找对话框是否已经存在。如果它不存在，DoShowModelessFindDialogBox将通过对话管理器创建一个新的对话框。它可以在新窗口记录的refCon域中存储代表查找对话框的常数，使该窗口可视，并绘制对话框常数。如果查找对话框已经存在，那么DoShowModelessFindDialogBox使该对话框窗口可视并选取它。当窗口管理器生成一个激活事件时，应用程序将调用其自己的过程来绘制此项内容。

窗口管理器参考

本节介绍窗口管理器的数据结构和例程。此外列出了窗口管理器使用的资源，并介绍了窗口（'WIND'）和窗口彩色表（'wctb'）资源。

数据结构

本节介绍窗口管理器的数据结构：窗口记录、彩色窗口记录、状态数据记录、窗口彩色表记录、辅助窗口记录和窗口列表。

一个窗口记录或彩色窗口记录说明一个单独的窗口。它包含用于显示窗口的图形端口记录。

状态数据记录存储两种矩形，即用户状态和标准状态，它是按照用户和您的应用程序的指定来定义窗口的大小和位置。当用户单击缩放格时，您的应用程序可以在这两种状态之间进行交换。

窗口彩色表定义的彩色将用于绘制窗口的框架和醒目显示所选定的文字。通常您使用系统设定的窗口彩色表，用户可通过彩色控制板选定彩色以产生窗口。如果您的应用程序需要专门控制框架的彩色，您也可以设定您自己的窗口彩色表。

窗口管理器使用辅助窗口记录可将窗口与其窗口彩色表相关联。窗口管理器使用窗口列表来追踪桌面上的所有窗口。

彩色窗口记录

窗口管理器为桌面上的每个窗口保留一个窗口记录或彩色窗口记录。

窗口管理器提供的例程可令您在需要时随时存取窗口记录。您的应用程序通常不能直接更改窗口记录中的域。

CWindowRecord数据类型可为一个彩色窗口定义窗口记录。CWindowPeek数据类型是指向一个彩色窗口记录的指针。在窗口记录中的第一个域实际上是说明窗口的图形端口记录。CWindowPtr 数据类型定义成一个指向窗口的图形端口的指针。

当Color QuickDraw 无效时， 您可以使用在下一节“窗口记录”中介绍的并行数据类型 WindowRecord、WindowPeek和WindowPtr来创建单色窗口。

考虑到兼容性，WindowPtr 和WindowPeek数据类型可以指向一个彩色窗口记录或单色窗口记录。您可以使用WindowPtr 数据类型在多数窗口管理器例程中指定一个窗口，也可以用该数据类型在QuickDraw 例程中指定一个图形端口，该例程采用GrafPtr数据类型 。请注意，您仅可以通过WindowPtr和 CWindowPtr数据类型存取用于窗口图形端口的那些域，而不能存取窗口记录的其余域。您可以在低级窗口管理器例程和您自己的例程中使用WindowPeek和 CWindowPeek 数据类型来存取非图形端口的窗口记录域。

用于操纵彩色窗口的例程可以从窗口彩色表和辅助窗口记录获取彩色信息，请参阅本节第4 － 71页中“窗口彩色表记录”和第4 － 73页中“辅助窗口记录”的介绍。

```
TYPE      CWindowPtr     =   ^CGrafPtr;
          CWindowPeek    =   ^CWindowRecord;

TYPE      CWindowRecord =
    RECORD
          port:          CGrafPort;      {window's graphics port}
          windowKind:    Integer;        {class of the window}
          visible:       Boolean;        {visibility}
          hilited:       Boolean;        {highlighting}
          goAwayFlag:    Boolean;        {presence of close box}
          spareFlag:     Boolean;        {presence of zoom box}
          strucRgn:      RgnHandle;      {handle to structure region}
          contRgn:       Rgandle;        {handle to content region}
          updateRgn:     RgnHandle;      {handle to update region}
          windowDefProc:Handle;          {handle to window definition}
                                         { function}
          dataHandle:    Handle;         {handle to window state}
                                         { data record}
          titleHandle:   StringHandle;   {handle to window title}
          titleWidth:    Integer;        {title width in pixels}
          controlList:   ControlHandle;{handle to control list}
          nextWindow:    CWindowPeek;    {pointer to next window}
                                         { record in window list}
          windowPic:     PicHandle;      {handle to optional picture}
          refCon:        LongInt;        {storage available to your}
                                         { application}
    END;
```

域说明

端口　　是说明图形端口的图形端口记录，该图形端口用于绘制窗口。在《Inside Macintpsh 绘图》(*Inside Macintpsh : Imaging*) 中说明的图形端口记录可以定义用于绘制的矩形、窗口的可视区域、窗口的剪贴区域和一组当前的绘制特征，如填充模式、画笔位置和画笔大小。

windowkind　　窗口类－即如何创建窗口。窗口管理器在创建窗口记录时填写此域。当窗口是由桌面附件创建时，它在 WindowKind 中放置负数值。(此值是桌面附件的参考标识符。)此域亦可以包含这两个常数之一:

```
CONST
    dialogKind    = 2;     {dialog or alert window}
    userKind      = 8;     {window created by an}
                           { application}
```

dialogKind值标识由系统软件创建的、或通过对话管理器间接地由您的应用程序创建的所有对话或警告栏窗口。对话管理器使用此域协助跟踪对话警告栏窗口。

userKind值代表直接由您的应用程序创建的一个窗口。

`visible` 是一个表示窗口是否可视的布尔值。如果窗口是可视的，窗口管理器将此域设定为 TRUE；反之为 FALSE 。可视性仅表明窗口是否可以显示出来，并不表示您是否能够在屏幕上看到它。(例如，完全被另一个窗口所覆盖的一 个窗口仍然是可视的，尽管用户无法在屏幕上看到它)。

`hilited` 是一个布尔值，表示是否醒目显示窗口—即在标题栏上绘制跑道线。一般醒目显示当前活动窗口时，hilited域中包含TRUE，反之为FALSE。

`goAwayFlag` 是一个布尔值，表示窗口是否具有一个关闭格。窗口管理器创建窗口根据'WIND'资源信息填写此域，或是根据传递给创建窗口的函数的参数填写此域。如果goAwayFlag 的值是TRUE，并且该窗口类型支持关闭格，那么一旦此窗口处于前台，窗口管理器将为其绘制关闭格。

`spareFlag` 是一个布尔值，表示窗口类型是否支持缩放。如果窗口的类型属于包括缩放框（zoomDocProc,zoomNoGrow，甚至mokalDBox-Proc+zoomDocProc）的一种，窗口管理器便设定此域为TRUE。

`strucRgn` 是指向结构区域的一个间接指针，在全局坐标中定义。由窗口所覆盖的整个屏幕区域叫结构区域。它包括窗口内容和窗口框架两部分。

`contRgn` 是用于内容区域的一个间接指针，在全局坐标中定义。内容区域是窗口的一部分，其中包含文稿、对话或其它数据；或是窗口控制部件；或是尺寸格。

`updateRgn` 是用于内容区域的一个间接指针，在全局坐标中定义。更新区域是必须要刷新的窗口部分。它是由窗口管理器和您的应用程序共同保留。更新区域中不包括被其它窗口所覆盖的窗口部分。

windowDeProc 是用于控制窗口的定义函数的一个间接指针。您的应用程序不必直接存取此域。在仅使用24位寻址的Macintosh方式中，此域包含用于窗口定义函数的一个间接指针和窗口变量代码。不论是何种寻址方式，如果您需要知道变量代码，那么请调用GetVariant函数。

dataHandle 通常是用于窗口定义函数的数据区域的一个间接指针。对于可缩放窗口， dataHandle 包含一个用于 WStateData 记录的间接指针，其中包含用户状态和标准状态矩形。从第4-70页的“窗口状态数据记录”中介绍了 WStateData 记录。

对于仅需要使用4 字节数据的窗口定义函数可以直接使用dataHandle 域，不必将间接指针存储到该域中。例如处理圆角窗口的窗口定义函数将曲线的直径存储在 dataHandle 域中。

titleHandle 是一个用于定义窗口标题的字符串的间接指针。

titleWidth 以像素计算的窗口标题宽度。

controlList 是由控制管理器使用的窗口控制列表的一个间接指针 请参阅本书的“控制管理器”中有关控制列表的说明）

nextWindow 是指向窗口列表中的下一个窗口的指针，即在桌面上此窗口后面的一个窗口。在桌面上最后一个窗口的窗口记录中，nextWindow域设定为NIL。

windowPic 是用于窗口内容的QuickDraw图片的一个间接指针。窗口管理器初始设定windowPic域为NIL 。如果您希望使用该窗口显示稳定的图象， 那么可以使用setWindowPic过程将用于该图片的间接指针置于此域中。当窗口的内容需要更新时， 窗口管理器便刷新内容本身， 而不是生成一个更新事件。

refCon 即窗口的参考值域， 它只是您的应用程序可随意使用的存储空间。通过将窗口类型常数存储在警告和对话窗口记录的 refCon 域中，将用于文稿记录的间接指针存储在文稿窗口记录的 refCon 域中，本章的示例代码就可以利用refcon 域使窗口与它所显示的数据相关联。

附 注

窗口记录中不包含关闭格、拖拽区域、缩放格和尺寸格，因为它们不需要形式数据结构用于 QuickDraw 中所定义的区域。窗口定义函数确定这些区域的位置。◆

窗口记录

如果彩色 QuickDraw 是无效的，那么您可利用并行数据结构（窗口记录）来创建窗口。彩色窗口记录与窗口记录的唯一不同之处是彩色窗口记录指向彩色图形端口，可以充分利用 Macintosh 电脑的彩色功能，而窗口记录指向单色图形端口。

用于说明窗口记录的数据类型 WindowRecord、WindowPtr 和 WindowPeek 与用于说明彩色窗口记录的数据类型是并行的，在单色窗口记录中的域与彩色窗口记录中的域是相同的。有关的详尽说明，请参阅从第 4 – 65 页的“彩色窗口记录”。

```
TYPE    WindowPtr       =   ^GrafPtr;
        WindowPeek      =   ^WindowRecord;

TYPE    WindowRecord    =               {all fields have same use}
  RECORD                                { as in color window record}
        port:          CGrafPort;       {window's graphics port}
        windowKind:    Integer;         {class of the window}
        visible:       Boolean;         {visibility}
        hilited:       Boolean;         {highlighting}
        goAwayFlag:    Boolean;         {presence of close box}
        spareFlag:     Boolean;         {presence of zoom box}
        strucRgn:      RgnHandle;       {handle to structure region}
        contRgn:       Rgandle;         {handle to content region}
        updateRgn:     RgnHandle;       {handle to update region}
        windowDefProc:Handle;           {handle to window definition}
                                        { function}
        dataHandle:    Handle;          {handle to window state}
                                        { data record}
        titleHandle:   StringHandle;    {handle to window title}
        titleWidth:    Integer;         {title width in pixels}
        controlList:   ControlHandle;   {handle to control list}
        nextWindow:    CWindowPeek;     {pointer to next window}
                                        { record in window list}
        windowPic:     PicHandle;       {handle to optional picture}
        refCon:        LongInt;         {storage available to your}
                                        { application}
  END;
```

窗口状态数据记录

缩放格允许用户在两种窗口位置和大小之间快速地切换用户状态和标准状态。在窗口状态数据记录中，窗口管理器将用户状态存储起来，而您的应用程序将标准状态存储起来，窗口状态数据记录的间接指针存在于窗口记录的 dataHandle 域中。

WStateData 记录数据类型可定义窗口状态数据记录。

```
TYPE   WStateDataPtr    = ^WStateData;
       WStateDataHandle   = ^WStateDataPtr;

       WStateData  =
       RECORD
            userState: Rect;  {size and location estabished by user}
            StdState: Rect;   {size and location established by app}
       END;
```

域说明

userState　是一个矩形，用于说明用户建立的窗口大小和位置。窗口管理器将用户状态初始化成首次显示时的窗口大小和位置。然后每当用户调整窗口大小时便更新 userState 域。
虽然用户状态可指定窗口的大小和位置，但窗口管理器仅当用户调整窗口大小时才更新状态数据记录，而不是当用户只移动窗口时。

stdState　是用于说明窗口大小和位置的矩形，考虑到文稿的功能、屏幕的可用空间、处于用户状态的窗口位置这些因素，此窗口大小和位置应是您的应用程序认为最方便的。如果您的应用程序没有定义标准状态，窗口管理器便自动地将主屏幕上的整个灰色区域设定为标准状态，同时边界的所有边减去三个象素。用户不能更改窗口的标准状态。当用户每次缩放到标准状态时，您的应用程序一般要计算并设定标准状态。例如，在字处理应用程序中，如果可能的话，标准状态的窗口可能显示整页，或者完整宽度的一页，其长度随屏幕而定，但尽量最长。如果用户通过页面设置来更改页的大小，应用程序可能会调整标准状态，以便适应新的页面大小。(请参阅《Macintosh 用户界面设计原则》(*Macintosh Human Interface Guidelines*) 中关于您的应用程序如何确定打开和缩放窗口的详尽说明。)

ZoomWIndow　过程根据窗口状态数据记录中的值来更改窗口的大小。当用户缩放 “in” 时该过程将窗口更改成用户状态；当用户缩放 “out” 时，更改成标准状态。有关缩放窗口的详尽说明，请参阅页 4-53 页的 “缩放窗口”。有关缩放窗口时您将调用的例程的说明，请参阅第 4-101 页的 “缩放窗口”。

窗口彩色表记录

用户可通过彩色控制板来控制窗口框架和醒目显示文字所用的色彩。一般来说，您的应用程序不能超越用户选择的色彩，这些色彩是存储在系统设定窗口彩色表中。如果您需要专门的控制窗口的色彩，那么可以为您的应用程序的窗口定义窗口彩色表。

窗口管理器在 WinCTab 类型的数据结构中可保留彩色信息表。

您可以定义您自己的窗口彩色表，并通过 SetWinColor 过程将其应用于现有窗口中。

当您创建一个窗口时，若要建立用于该窗口的窗口彩色表，您提供的窗口彩色('wctb')资源的资源标识符应与定义该窗口的'WIND'资源的资源标识符 相同。

WCTabPtr 数据类型是一个指向窗口彩色表记录的指针，WTabHandle 是用于窗口彩色表记录的一个间接指针。

```
TYPE    WCTabPtr    = ^WinCTab;
        WCTabHandle  =^WCTabPtr;



TYPE    WinCTab   =
        RECORD
            wCSeed:        LongInt;      {reserved}
            wCReserveed:   Integer;      {reserved}
            ctSize:        Integer;      {number of entries in table -1}
            ctTable:       ARRAY[0..4] OF ColorSpec;
                                         {array of color specification}
                                         { records}
        END;
```

域说明

wCSeed　被保留。

wCReserved　被保留。

ctSize　表中输入项数目减去1。如果您想用标准窗口定义函数建立一个彩色表以备使用，那么此域的最大值是12。自定窗口定义函数可以使用任意大小的彩色表。

`ctTable` 是colorSpec记录的数组。在一个窗口彩色表中，每个colorSpec记录在第一个字中指定窗口部分，在另外三个字中指定 RGB 值:

colorSpec记录的数值域可指定一个常数用于定义窗口的哪一部分由色彩控制。对于标准窗口定义函数所使用的窗口彩色表，您可以指定具有这些含义的下述数值。

```
TYPE ColorSpec =
     RECORD
        value:    Integer;      {part identifier}
        rgb:      RGBColor;     {RGB value}
     END
```

```
CONST
wContentColor      =0;  {content region background}
wFrameColor        =1;  {window outline}
wTextColor         =2;  {window title and button}
                        { text}
wHiliteColor       =3;  {reserved}
wTitleBarColor     =4;  {reserved}
wHiliteColorLight  =5;  {lightest stripes in}
                        { title bar and lightest}
                        { dimmed text}
wHiliteColorDark   =6;  {darlest stro[es om}
                        { tot;e bar amd}
                        { darlest do,,ed}
                        { text}
wTitleBarLight     =7;  {lightest parts of}
                        { title bar background}
wTitleBarDark      =8;  {darkest parts of}
                        { title bar background}
wDialogLight       =9;  {lightest element}
                        { of dialog box frame}
wDialogDark        =10; [darkest element of}
                        { dialog box frame}
wTingeLight        =11; {lightest window tinging}
wTingeDark         =12; {darkest window tinging}
```

附 注

在 System 5 和 System 6 中的部分代码与这里介绍的部分代码(仅适用于System 7)有很大的区别。◆

窗口部分可以以任意顺序出现在表中。

colorSpec 记录的 rgb 域中包含三个数据字，可以指定所用彩色的红、绿和蓝值。在《Inside Macintosh: Imaging》中介绍了 RGBColor 数据类型的定义。

当您的应用程序创建一个窗口时，窗口管理器首先寻找具有相同资源标识符 的'wctb'类型资源作为该窗口的'WIND'资源。如果能找到，就可利用该资源中的信息创建该窗口的窗口彩色表，然后用这些色彩显示该窗口。如果没有找到具有相同资源标识符的窗口色彩表资源作为您的窗口资源，窗口管理器便使用系统设定的系统窗口彩色表，在应用程序启动期间读到堆中。

在创建窗口之后，您可以使用 SetWinColor 过程来更改窗口的窗口彩色表中的输入项，该过程在第 4-114 页介绍。

请参阅第 4-127 上的“窗口彩色表资源”中关于窗口色彩表资源的说明。

辅助窗口记录

辅助窗口记录可以指定窗口所使用的彩色表，其中包含由对话管理器和窗口管理器使用的参考信息。

窗口管理器创建并保留一个辅助窗口记录中的信息；您的应用程序通常不(如果有的话）需要存取一个辅助窗口记录。

```
TYPE    AuxWinPtr          = ^AuxWinRec;
        AuxWinHandle       = ^AuxWinPtr;

        AuxwinRec =
        RECORD
            awNext:        AuxWinHandle; {handle to next record}
            awOwner:       WindowPtr;    {pointer to window}
                                         { asociated with this}
                                         { record}
            awCTable:      CTabHandle;   {handle to color table}
            dialogCItem:   Handle;       {storage used by }
                                         { Dialog Manager}
            awFlags:       LongInt;      {reference constant,}
                                         { for application's use}
        END;
```

域说明

awNext　是用于辅助窗口列表中下一个记录的间接指针，窗口管理器用此间接指针把辅助窗口列表作为连接起来的列表加以保留。如果一个窗口使用系统设定的辅助窗口记录，则此值为 NIL。

awOwner　是指向使用此记录的窗口的一个指针。系统设定的辅助窗口记录的 awOwner 域设定为 NIL。

awCTable　是用于窗口的彩色表的间接指针。除非您另外指定，此间接指针将用于系统窗口色彩表。

dialogCItem 供对话管理器使用的专用存储器。

awFlags　保留。

awReserved 保留。

awRefCon　参考常数，应用程序一般用其将辅助窗口记录与文稿记录相关联。

除特殊情况外，您的应用程序不需要操纵窗口色彩表或辅助窗口记录。

为了在共享环境中与其它应用程序保持兼容性，您的应用程序不应直接操纵系统彩色表，而应该查阅《Inside Macintosh 绘图》(*Inside Macintosh: Imaging*) 中关于调色板管理器的说明。如果您的应用程序提供其本身的窗口和控制部件定义函数，这些函数使用用户桌面彩色选择的方式应该与标准窗口和控制部件定义函数的方式相同。

窗口列表

窗口管理器可以在称为 WindowList 的专用结构中保留桌面上窗口的信息。窗口列表中包含指向桌面上（包括可视和不可视的窗口）所有窗口的指针，列表中还包含窗口管理器用来保留桌面的其它信息。

您的应用程序不应该在窗口列表中直接存取信息。窗口列表的结构是窗口管理器专用的。

全局变量 WindowList 中包含指向窗口列表中第一个窗口的指针。

窗口管理器例程

本节介绍用于创建、显示和管理窗口的全部例程集。

初始化窗口管理器

您必须调用 InitWindows 过程来初始化窗口管理器之后，才可以使用窗口管理器的其它例程。

作为初始化的一部分，InitWindows 可创建窗口管理器端口，它是占据所有的主屏幕的图形端口。在配备了 Color QuickDraw 的 Macintosh 电脑中，窗口管理端口命名为 WMgrCPort，在仅配有 QuickDraw 的电脑中，该端口命名为 WMgrPort。

通常，您的应用程序并不需要知道窗口管理器端口。但如果有必要，您可以通过调用过程 GetWMgrPort 或 GetCWMgrPort 检索它的指针。您的应用程序不应直接在窗口管理器端口绘图，通过自定窗口定义函数除外。

窗口管理器在窗口管理器端口中绘制您的应用程序窗口。窗口管理器端口的端口矩形是主屏幕（screenBits.bounds）的界定矩形，为了适应配有多个显示器的系统，QuickDraw 把screeBits.bounds端口矩形作为一种特殊情况识别，并允许在桌面任意部分上绘图。

InitWindows

过程 InitWindows 为您的初始化应用程序窗口管理器。在调用 InitWindows 之前，您必须调用 InitGraf 和 InitFonts 过程来初始化 QuickDraw 和字体管理器，在《Inside Macintosh 绘图》(*Inside Macintosh: Imaging*) 和《Inside Macintosh 文本》(*Inside Macintosh : Text*) 中有相关的说明。

```
PROCEDURE InitWindows;
```

说明

InitWindows 过程初始化窗口管理器。

汇编语言信息

桌面在初始化后需要刷新时，窗口管理器检查全局变量 DeskHook，该变量可作为应用程序定义的用来绘制桌面的例程的指针。系统启动后该变量通常设定为0。如果您用启动期间将被执行的代码段显示窗口，则应把 DeskHook 设定为 0。注意，在 System 6 以后的软件版本中，并不保证窗口管理器全局变量的使用是兼容的。

创建窗口

您可用两种方法创建窗口：

- 依据窗口资源（'WIND' 类型的资源），使用 GetNewCWindow 和 GetNewWindow 函数
- 依据作为参数传递给 NewCWindow 和 NewWindow 函数所有窗口特性

依据资源创建窗口允许您针对不同的语言本地化您的应用程序，允许您在应用程序开发期间通过只改变窗口资源来改变您的窗口的特性。

所有这4个函数，GetNewCwindow，GetNewWindow，NewCWindow和NewWindow都可在您应用程序堆中为新窗口的窗口记录分配存储器空间。若想对存储器的使用进行更多的控制，在创建窗口时您自己可以分配存储器空间并传递一个指针。在这两种情形中，窗口管理器填充数据结构并返回它的指针。

GetNewCWindow

使用GetNewCWindow函数可创建彩色窗口，窗口特征由具有一个指定资源标识符的'WIND'资源定义。

```
FUNCTION     GetNewCWindow (WindowID: Integer; wStorage: Ptr;
                            behind: WindowPtr): WindowPtr;
```

`WindowID` 定义窗口特征的 'WIND' 资源的资源标识符。

`wStorage` 窗口记录的存储器空间指针。

如果您给wStorage指定的值是NIL，GetNewCWindow函数就把窗口记录作为一个不可重定位对象在堆中为其分配空间。通过尽早在您的初始代码中为您的应用程序的窗口记录分配存储器，您可减少堆存储碎片的机会。不管您什么时候需要创建窗口，您都可从您自己的块中分配存储器并在wStorage参数中传递它的指针。

`behind` 桌面上立即出现在新窗口前面的窗口的指针。

若要在桌面上所有其它窗口的前面放置一个新窗口，指定一个值Pointer（-1）。当您在所有其它窗口前面放置一个窗口时，GetNewCWindow取消以前当前活动窗口的醒目显示，醒目显示最新创建的窗口，并生成相应的激活事件。注意，如果您在桌面上所有其它窗口的前面创建一个不可视窗口，用户要直到您使新窗口可视（或使另一个窗口成为现用）才能看到当前活动窗口。

若要把一个新窗口的放置在所有其它窗口的后面，指定为NIL。

说明

GetNewCWindow函数依据指定的窗口的资源创建一个新的彩色窗口并返回最新创建的窗口记录的指针。在大多数窗口管理器例程中您可利用返回的窗口指针访问该窗口。如果GetNewCWindow不能从资源文件中读取窗口或窗口定义函数，它就返回NIL。

GetNewCWindow 函数采用与 'WIND' 资源相同的资源标识符寻找 'wctb' 资源。如果它找到一个，它将根据 'wctb' 中的窗口彩色信息给窗口框架着色并醒目显示选定的文字。

如果窗口的定义函数（在窗口记录中指定）没在存储器中，GetNewCWindow 将把它读进存储器，并在窗口记录中储存它的间接指针。GetNewCWindow 在应用程序堆中为窗口的结构和内容区分配空间，并请求窗口定义函数计算这些区域。

为了创建窗口，GetNewCWindow 在窗口资源中检索窗口特性，然后把这些特性作为参数调用 NewCWindow 函数。

GetNewCWindow 函数在彩色图形端口中创建窗口。调用 GetNewCWindow 之前，需要证实 Color QuickDraw 是否可用。在调用 Gestalt 函数进行初始化期间，您的应用程序一般设置它自己的反映系统设置的全局变量。请参阅《Inside Macintosh 概述》(*Inside Macintosh : Overview*) 关于建立本地配置的进一步介绍。

特殊考虑

注意 GetNewCWindow 函数返回一个 WindowPtr 类型的值，而不是 CWindowPtr 类型。

如果您让窗口管理器在您的应用程序堆中创建窗口记录，就调用 DisposeWindow 来处理窗口的窗口记录。如果您自己为窗口记录分配存储器并给 GetNewCWindow 传递存储器指针，就使用过程 CloseWindow 来关闭窗口，调用过程 DisposePtr 来处理窗口记录，在《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中有关于这个过程的说明。

亦请参阅

请参阅 4–28 页的程序清单 4–3 中的实例，它调用 GetNewCWindow 依据窗口资源创建一个新窗口。

若想进一步了解关于窗口特性和窗口资源的情况，请参阅 4–79 页对 NewCWindow 的说明，和 4–124 页“窗口资源”一节中对 'WIND' 资源的说明。

若想了解用来关闭窗口和从存储器中取消数据结构的这些过程，请参阅 4–105 页对 DisposeWindow 过程的说明，4–104 页的 CloseWindow 过程，以及《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中的 DisposePtr 过程。请参阅 4–61 页的程序清单 4–17 关于关闭文稿窗口的实例。

GetNewWindow

当 Color QuickDraw 不可用时，使用 GetNewWindow 函数依据窗口资源创建新窗口。GetNewWindow 函数采用与 GetNewCWindow 相同的参数并返一个 WindowPtr类型的值。唯一的不同就是它创建一个单色图形端口，不是彩色图形端口。窗口记录与说明单色和彩色图形端口的图形端口记录大小相同，在大多数窗口管理器例程中可互换使用。

```
FUNCTION GetNewWindow (WindowID: Integer; wStorage: Ptr;
                       behind: WindowPtr): WindowPtr;
```

`WindowID` 定义窗口特征的 'WIND' 资源的资源标识符。

`wStorage` 窗口记录的存储器空间指针。

如果您给 wStorage 指定的值是 NIL，GetNewWindow 函数将把窗口记录作为一个不可重定位对象，在堆中为其分配空间，通过尽早在您的初始化代码中为您的应用程序的窗口记录分配存储器，您可减少堆破碎的机会。不管您什么时候需要创建窗口，您都可从您自己的块中分配存储器并在 wStorage 参数中传递它的指针。

`behind` 桌面上立即出现在新窗口前面的窗口的指针。

若要在桌面上所有其它窗口的前面放置一个新窗口，指定一个值 Pointer (-1)。当您在所有其它窗口前面放置一窗口时，GetNewWindow取消以前当前活动窗口的醒目显示，醒目显示最新创建的窗口，并生成相应的激活事件。注意，如果您在桌面上所有其它窗口前面创建一个不可视窗口，用户要直到您使新窗口可视（或使另一个窗口变成当前活动）才能看到当前活动窗口。

若要把一新窗口放置于所有其它窗口的后面，指定一个值 NIL。

说明

像 GetNewCWindow 一样，GetNewWindow 依据窗口资源创建新窗口，但它是创建一个单色窗口。GetNewWindow函数依据指定的窗口资源创建新窗口并返回最新创建的窗口记录的指针。在大多数窗口管理器例程中您可用返回的窗口指针访问这个窗口。如果 GetNewWindow 过程不能从资源文件中读取窗口或窗口定义函数，它就返回 NIL。

如果窗口的定义函数（在窗口记录中指定）不在存储器中，GetNewWindow 就把它读进存储器，并在窗口记录中储存它的间接指针。GetNewWindow 在应用程序堆中为窗口的结构和内容区分配空间，并请求窗口定义函数计算这些区域。

为了创建窗口，GetNewWindow 在窗口资源中检索窗口特性，然后调用 NewWindow 函数，把这些特性作为参数传递给 NewWindow 函数。

特殊考虑

如果您让窗口管理器在您的应用程序堆中创建窗口记录，就调用用 DisposeWindow 来处理窗口的窗口记录。如果您自己给窗口记录分配存储器并给 GetNewWindow 传递一个指针，就用过程 CloseWindow 来关闭窗口，用过程 DisposePtr 来处理窗口记录，在《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中有关于该过程的说明。

亦请参阅

若想进一步了解关于窗口特性和窗口资源的情况，请参阅 4-82 页对 NewWindow 的说明，和始于 4-124 页"窗口资源"一节中对 'WIND' 资源的说明。

若想了解用来关闭窗口和在存储器中取消结构的这些过程，请参阅 4-105 页关于 DisposeWindow 的说明，和 4-104 页的 CloseWindow 过程，以及《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中的 DisposePtr 过程。

NewCWindow

您可使用 NewCWindow 函数创建具有指定特性列表的窗口

```
FUNCTION NewCWindow(wStorage: Ptr; bowndsRect: Rect;
                    title: str255; visible: Boolean;
                    procID: Integer; behind: WindowPtr;
                    goAwayFlag: Boolean;
                    refCon: LongInt):WindowPtr;
```

`wStorage` 窗口记录指针。如果您给 wStorage 指定的值是 NIL，NewCWindow 就把窗口记录作为一个不可重定位对象在堆中为其分配空间。在您的应用程序专门保留的一块存储器中分配存储器，并在 wStorage 参数中传递它的指针，您可减少堆破碎出现的机会。

`boundsRect` 在全局坐标中的一个矩形，它指定窗口的初始大小和位置。这个参数就是窗口图形端口的端口矩形。对于标准的窗口类型，boundsRect 域定义窗口的内容区域。NewCWindow 函数将局部坐标系统的原点置于端口矩形的左上角。

附　注

NewCWindow 函数实际上调用QuickDraw过程OpenCPort来创建图形端口。位映像，笔模式，以及窗口的图形端口的其它特性与OpenCport设定的系统设定值相同，但字符字体除外，它设定成应用程序字体而非系统字体。◆

`title` 指定窗口标题的字符串

如果标题太长以致于标题栏容纳不下，标题将被截短。如果窗口有关闭格，则标题尾部的字符将被截去；如果没有关闭格，则以标题的中心为准则，从两头截去字符。

若要在有标题栏的窗口中禁止标题，可在title参数中传递一个空字符串，但不能传递NIL。

`visible` 表明可视性状态的布尔值：TRUE表示窗口管理器显示窗口；FALSE表示不显示。

如果 visible 参数的值为TRUE，窗口管理器在窗口一创建时就绘制新窗口。窗口管理器首先调用窗口定义函数绘制窗口的框架。如果goAwayFlag 参数的值也是 TRUE 并且窗口处在最前面（即如果 behind 参数的值为Pointer（-1）），窗口管理器就指导窗口定义函数在窗口框架中绘制一个关闭格。绘制完框架后，窗口管理器生成一个更新事件来触发您的应用程序绘制内容区域。

当您创建窗口时，您一般指定FALSE作为visible参数的值。当您准备显示窗口时，您可调用ShowWindow过程，在4-88页中有相关的说明。

`procID` 窗口定义标识符，指定窗口定义函数和这个定义函数内的变量代码。

窗口管理器支持9种标准窗口类型，它们由两个窗口定义函数处理。您可通过指定一个窗口定义标识符常数来创建标准类型的窗口：

```
CONST
   documentProc      =0;  {standard document }
                          { window, no zoom box}
   dBoxProc          =1;  {alert box or modal }
                          { dialog box}
   plainDBox         =2;  {plain box}
   altDBoxProc       =3;  {plain box with shadow}
   noGrowDocProc     =4;  {movable window, }
                          { no size box or zoom box}
   movableDBoxProc   =5;  {movable modal dialog box}
   zoomDocProc       =8;  {standard document window}
   zoomNoGrow        =12; {zoomable, nonresizable }
                          { window}
   rDocProc          =16; {rounded-corner window}
```

若想了解关于9种标准窗口类型的说明，请参阅4－8页的“窗口类型”。

通过把一个整数添加至rDocProc常数，您可控制圆角窗口曲率，4-124页的“窗口资源”对此进行了说明。

behind　桌面上立即出现在新窗口前面的窗口的指针

若要在桌面上其它所有窗口的前面放置一个新窗口，指定一个值Pointer(-1)。当您在所有其它窗口前面放置一个新窗口时，NewCWindow取消以前当前活动窗口的醒目显示，醒目显示最新创建的窗口，并生成激活事件触发您的应用程序更新这两个窗口。注意，如果您在桌面上所有其它窗口前面创建一个不可视窗口，用户要直到您使新窗口可视 或使别一个窗口变成现用）才能看到当前活动窗口。

若要把一个新窗口放置于所有其它窗口的后面，指定一个值NIL。

goAwayFlag　确定窗口是否有关闭格的布尔值。如果goAwayFlag的值为TRUE并且窗口类型支持关闭格，窗口管理器就在标题栏中绘制一个关闭格并且识别在关闭区域内的鼠标点按；如果goAwayFlag的值为FALSE或者窗口类型不支持关闭格，窗口管理器就不做上述工作。

refCon　窗口的参考常数，仅由您的应用程序设定和使用。 请参阅始于4-23页的“管理多窗口”，其中提出了一些使用refCon参数的方法。）

说明

NewCWindow 函数根据其参数的指定创建一个窗口，把它添加至窗口列表，并返回最新创建的窗口记录的指针。在大多数窗口管理器例程中您可用返回的窗口指针访问这个窗口。如果NewCWindow不能从资源文件中读窗口定义函数，它就返回NIL。

NewCWindow函数用与 `WIND’ 资源相同的资源标识符寻找 `wctb’ 资源。如果它找到一个，它就根据 `wctb’ 资源中的窗口彩色信息给窗口框架上色并醒目显示。

如果窗口的定义函数不在存储器中，NewCWindow就把它读进存储器并在窗口记录中存储它的间接指针。NewCWindow为窗口的结构和内容区域分配空间并请求窗口定义函数计算这些区域。

把您的窗口的特性作为资源存储起来，特别是窗口标题的窗口项目，将使您的应用程序能更容易实现本地化。

NewCWindow函数在彩色图形端口创建窗口。尽可能地创建彩色窗口，保证您的窗口能以用户选定的任何彩色选项出现在彩色显示器上。调用 GetNewCWindow 之前，需证

实 color QuickDraw 可用。在调用 Gestalt 函数进行初始化期间，您的应用程序一般设置一组它自己的反映系统设置的全局变量。请参阅《Inside Macintosh概述》(*Inside Macintosh : Overview*) 一章关于建立本地配置的进一步介绍。

注意，函数 NewCWindow 返回一个 WindowPtr 类型的值，不是 CWindowPtr。

特殊考虑

如果您希望窗口管理器在您的应用程序堆中创建窗口记录，就调用 DisposeWindow过程来关闭窗口和处理窗口记录。如果您自己为窗口记录分配存储器并给 NewCWindow 传递一个指针，就使用 CloseWindow 过程来关闭窗口，用 DisposePtr 过程来处理窗口记录，在《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中有关于该过程的说明。

亦请参阅

若想了解用来关闭窗口和在存储器中取消数据结构的这些过程，请参阅4-105页关于 DisposeWindow 过程的说明，和 4-104 页的 CloseWindow 过程，以及《Inside Macintosh 内存》(*Inside Macintosh : Memory*) 中的 DisposePtr 过程。

NewWindow

当 Color QuickDraw 不可用时，使用 NewWindow 函数来创建新窗口，窗口的特性由一个参数列表指定。NewWindow 函数采用与 NewCWindow 相同的参数，与 NewCWindow 类似，它返回一个 WindowPtr 类型的值作为它的函数结果。唯一的不同之处是 NewWindow 在单色图形端口中创建窗口，而不是在彩色图形端口中。说明单色和彩色图形端口的窗口记录和图形端口记录大小相同，在大多数窗口管理器例程中它们可以互换使用。

```
FUNCTION NewWindow (wStorage:Ptr;boundsRect:Rect;
                    title:Str255;visible:Boolean;
                    theProc:Integer;behind:WindowPtr;
                    goAwayFlag:Boolear;
                    refCon:LongInt):WindowPtr;
```

`wStorage` 窗口记录指针。如果您指定 NIL 作为 wStorage 的值，NewWindow 就把窗口记录作为一个不可重定位对象在堆中为其分配空间。在您的应用程序专门保留的一块存储器中分配存储器并在wStorage参数中传递它的指针，您可减少堆存储碎片出现的机会。

boundsRect 在全局坐标中的一个矩形，它指定窗口的初始大小和位置。这个参数也就是窗口图形端口矩形，对于标准的窗口类型，boundsRect 定义窗口的内容区域。NewWindow 函数将局部坐标系统的原点置于端口矩形的左上角。

附 注

NewWindow函数实际上调用QuickDraw过程OpenPort来创建图形端口。位映象，笔模式，以及窗口的图形端口的其它特性与OpenPort设定的系统设定值相同，但字符字体除外，它设定成应用程序字体而非系统字体。图形端口的端口边界和可视区域的坐标随着其端口矩形的改变而改变。

title 指定窗口标题的字符串。

如果标题太长标题栏容纳不下，标题将被截短。如果窗口有关闭格，则标题尾部的字符将被截去；如果没有关闭格，则以标题的中心为准则，从两头截去字符。

若要在有标题栏的窗口中禁止标题，可在title参数中传递一个空字符串，不能传递 NIL。

Visible 表明可视性状态的布尔值：TRUE 表示窗口管理器显示窗口；FALSE 表示不显示。

如果 Visible 参数的值为 TRUE，窗口管理器在窗口一存在时就绘制新窗口。窗口管理器首先调用窗口定义函数绘制窗口框架。如果goAwayFlag参数（下面将介绍）的值也是TRUE并且窗口处在最前面（即如果behind参数的值为Pointer(-1)），窗口管理器就指导窗口定义函数在窗口框架中绘制一个关闭格。绘制完框架后，窗口管理器生成一个更新事件来触发您的应用程序绘制内容区。

当您创建窗口时，您一般指定FALSE作为Visible参数的值，为您准备显示窗口时，您可调用ShowWindow过程来显示窗口，在4-88页中有相关的说明。

theProc 窗口定义标识符，指定窗口定义函数和这个定义函数的变量代码。

窗口管理器支持9种标准窗口类型，它们由两个窗口定义函数处理。您可通过指定一个类型常数来创建标准类型的窗口。

```
CONST
 dicumentProc  = 0;    {standard document }
                       { window, no zoom box}
 dBoxProc      = 1;    {alert box or modal }
                       { dialog box}
 plainDBox     = 2;    {plain box }
```

```
altDBoxProc    = 3;     {plain box with shadow}
noGrowDocProc = 4;      {movable window, }
                        { no size box or zoom box}
movableDBoxProc= 5;     {movable modal dialog box}
zoomDocProc    = 8;     {standard document window}
zoomNoGrow     = 12;    {zoomable, nonresizable }
                        { window}
rDocProc       = 16;    {rounded-corner window}
```

通过把一个整数添加至 rDocProc 常数，您可控制圆角窗口弯曲部分的直径，4-124 页的“窗口资源”对此进行了说明。

behind　桌面上立即出现在新窗口前面的窗口的指针。

若要在桌面上其它所有窗口的前面放置一个新窗口，将该参数值指定为 Pointer (-1)。当您在所有其它窗口前面放置一个新窗口时，NewWindow 取消以前当前活动窗口的醒目显示，醒目显示最新创建的窗口，并生成激活事件触发您的应用程序更新这两个窗口。注意，如果您在桌面上所有其它窗口前面创建一个不可视窗口，用户要直到您使新窗口可视（或使另一个窗口变成当前窗口）才能看到当前窗口。

若要把一个新窗口放置于所有其它窗口的后面，指定该参数值为NIL。

goAwayFlag　确定窗口是否有关闭格的布尔值。如果goAwayFlag的值为TRUE并且窗口类型支持关闭格，窗口管理器就在标题栏中绘制一个关闭格并且识别在关闭区域内的鼠标单击；如果goAwayFlag的值为FALSE或者窗口类型不支持关闭格，窗口管理器就不做上述工作。

refCon　窗口的参考常数，仅由您的应用程序设定和使用。(请参阅 4-23 页的“管理多窗口”，其中提出了一些使用 refCon 参数的方法。)

说明

NewWindow 函数根据其参数的指定创建一个窗口，把它添加至窗口列表，并返回最新创建的窗口记录的指针。在大多数窗口管理器例程中您可用返回的窗口指针访问这个窗口。如果NewWindow不能从资源文件中读取窗口定义函数，它就返回NIL。

如果窗口的定义函数不在存储器中，NewWindow 就把它读进存储器并在窗口记录中存储它的间接指针。NewWindow 为窗口的结构和内容区域分配空间并请求窗口定义函数计算这些区域。

把您的窗口的特性作为资源存储起来，特别是窗口标题和窗口项目，将使您的应用程序能更容易地实现本地化。

特殊考虑

如果您让窗口管理器在您的应用程序堆中创建窗口记录，就调用Dispose过程来关闭窗口并处理它的窗口记录。如果您自己为窗口记录分配存储器并给NewCWindow传递一个指针，就使用CloseWindow过程来关闭窗口，用DisposePtr过程来处理窗口记录，在《Inside Macintosh内存》(*Inside Macintosh : Memory*) 中有关于该过程的说明。

亦请参阅

若想了解用来关闭窗口和在存储器中取消数据结构的这些过程，请参阅4-105页关于DisposeWindow过程的说明，和4-104页的CloseWindow过程，以及《Inside Macintosh内存》(*Inside Macintosh : Memory*) 中的DisposePtr过程。

命名窗口

本节介绍设定和检索窗口标题的过程。

SetWTitle

用SetWTitle 过程来改变窗口的标题。

```
PROCEDURE   SetWTItle (theWindow:WindowPtr;title:Str255);
```

theWindow	窗口的窗口记录指针。
title	新的窗口标题。

说明

SetWTitle过程将窗口的标题改为指定的字符串，既在窗口记录中作修改又在屏幕上改变标题，并对窗口框架作必要的刷新。

当用户打开一个以前存储的文稿时，您一般创建一个新的(不可视的)窗口，窗口的标题未标出，然后调用SetWTitle，在显示窗口前将文稿的名称赋予该窗口。当用户以一个新的名称存储文稿时您也需调用SetWTitle。

若要在有标题栏的窗口中禁止标题时，传递一个空串，不能传递NIL。

不要在窗口的窗口记录中直接修改标题，只能通过 SetWTitle 过程修改标题。

GetWTitle

用 GetWTitle 过程来检索窗口的标题。

```
PROCEDURE GetWTitle (theWindow:WindowPtr; VAR title:Str255);
```

theWindow　窗口记录指针。

title　窗口标题。

说明

GetWTitle 过程通过 title 参数中返回窗口的标题。

您的应用程序很少需要确定一个窗口的标题。但在涉及到多个文件的操作中，当展示用户对话框时，很可能有这种需要。例如，一个“拼写检查”命令可能显示一个对话框，让用户从当前打开的所有文稿中选定要检查的文稿。

当您需要检索一个窗口的标题时，你应该采用 GetWTitle 过程，而不应从窗口的窗口记录中读标题。

显示窗口

本节介绍的窗口管理器例程可改变一个窗口的显示和它在窗口列表中的位置，但不改变窗口的大小和窗口在桌面上的位置。注意，窗口管理器自动地在屏幕上绘制所有可视窗口。

您的应用程序一般仅使用本节介绍的少数几个例程：DrawGrowIcon，SelectWindow，ShowWindow，以及偶尔使用的 HideWindow。

DrawGrowIcon

用 DrawGrowIcon 过程绘制窗口的尺寸格。

```
PROCEDURE DrawGrowIcon (theWindow: WindowPtr);
```

theWindow　窗口记录指针。

说明

DrawGrowIcon 过程绘制窗口的尺寸格，如果窗口不能调整大小就绘制相应的其它图像。当绘制包含尺寸格的窗口的内容区域时，您可调用 DrawGrowIcon。

图像的实际外观和准确位置取决于窗口类型和窗口的当前状态。DrawGrowIcon 自动地核查窗口的类型和状态并绘制相应的图像。

在一个当前活动的文稿窗口中，DrawGrowIcon 在窗口的图形端口矩形的右下角的尺寸格中绘制增长图像，它还绘制尺寸格和滚动杆区域的分界线。若要绘制尺寸格但不绘制滚动杆轮廓线时，可将窗口图形端口的 clipRgn 域设定成在窗口右下角的一个 15 × 15 像素矩形。

DrawGrowIcon 过程并不抹去滚动杆区域，因而如果您使用 DrawGrowIcon 绘制尺寸格和滚动杆轮廓线，当窗口大小改变时，即使窗口并不包含滚动杆，您也应该自己抹去这些区域。

在非现用文稿窗口中，DrawGrowIcon 绘制尺寸格和滚动杆区域的分界线并抹去尺寸格。

亦请参阅

请参阅 4-39 页的程序清单 4-8 关于绘制窗口的内容区域（包括尺寸格）的实例。请参阅 4-51 页的程序清单 4-11 关于窗口切换到后台时调用 DrawGrowIcon 取消尺寸格图标的实例。

SelectWindow

用 SelectWindow 过程使一个窗口变成当前活动的窗口。SelectWindow 过程改变窗口的现用状态但并不影响它的可视性。

```
PROCEDURE SelectWindow (theWindow: WindowPtr);
```

`theWindow`　窗口的窗口记录指针。

说明

SelectWindow 过程取消以前当前活动窗口的醒目显示，把指定的窗口送到前面，醒目显示这个窗口，并生成激活事件停用以前的活动窗口和激活指定的窗口。如果指定的窗口已经是当前活动窗口，SelectWindow 就不起任何作用。

即使指定的窗口不可视，SelectWindow 仍把窗口送到前面，激活该窗口，并停用以前的当前活动窗口。注意，在这种情况下屏幕上没有可视的当前活动窗口。如果您选定了一个不可视窗口，一定要立即调用 ShowWindow 使该窗口可视（使用户能看到）。

当光标在后台窗口的内容区域、用户按下鼠标按钮时，调用 SelectWindow 过程。

亦请参阅

请参阅4-44页程序清单4-9中的实例，当光标在后台窗口用户按下鼠标按钮时它调用 SelectWindow 来改变当前活动窗口。

请参阅4-64页程序清单4-18中的实例，在使用HideWindow使窗口不可视之后，需用 SelectWindow 和 ShowWindow 一起来还原窗口的活动、可视状态。

ShowWindow

调用 ShowWindow 过程使不可视窗口可视。

```
PROCEDURE ShowWindow (theWindow: WindowPtr);
```

theWindow　窗口的窗口记录指针。

说明

ShowWindow 过程使不可视窗可视。如果指定的窗口已经是可视的，ShowWindow就不起任何作用。您的应用程序一般以不可视状态创建一个新窗口，在完成对内容区域执行一些必要的设置后，才调用 ShowWindow 使窗口可视。

当您调用 ShowWindow 显示一个以前不可视的窗口时，窗口管理器绘制窗口框架并随后生成一个更新事件触发您的应用程序绘制内容区域。

如果最新可视的窗口是最前面的窗口，若这个窗口还没被醒目显示，ShowWindow 过程就醒目显示这个窗口，ShowWindow 还生成一个激活事件使窗口成为当前活动窗口。ShowWindow 过程并不激活桌面上不是最前面的窗口。

附　注

因为ShowWindow并不改变窗口叠放的前后次序，所以它不是HideWindow的逆过程。如果您用HideWindow使最前面的窗口不可视，并且HideWindow已经激活了另一个窗口，您就必须同时调用ShowWindow和SelectWindow使原来的窗口恢复到前面。

亦请参阅

请参阅4-60页程序清单4-16关于用户关闭对话框时暂时隐藏对话框窗口的实例。请参阅 4-64 页程序清单 4-18 关于随后调用 ShowWindow 重新显示窗口的实例。

HideWindow

调用 HideWindow 过程使一个窗口不可视。

```
PROCEDURE HideWindow (theWindow: WindowPtr);
```

theWindow　窗口的窗口记录指针。

说明

HideWindow过程使可视窗口变成不可视。如果您隐藏最前面的窗口，HideWindow 就取消这个窗口的醒目显示，把这个窗口之后的窗口送到前面，醒目显示新的最前面的窗口，并生成相应的激活事件。

若要完成 HideWindow 动作的逆动作，您必须同时调用 ShowWindow 和 SelectWindow，调用 ShowWindow 使窗口可视，调用 SelectWindow 来选定窗口。

亦请参阅

请参阅 4-60 页程序清单 4-16 关于用户关闭对话框时调用 HideWindow 暂时隐藏对话框窗口的实例。请参阅 4-64 页程序清单 4-18 关于随后重新显示窗口的配套实例。

ShowHide

调用 ShowHide 过程设定窗口的可视性状态。

```
PROCEDURE ShowHide(theWindow: WindowPtr; ShowFlag: Boolear);
```

theWindow　窗口的窗口记录指针。

ShowFlag　确定可视性状态的布尔值：TRUE 使窗口可视；FALSE 使窗口不可视。

说明

ShowHide 过程将窗口的可视性设定成 ShowFlag 参数指定的状态。如果 ShowFlag 的值是 TRUE　若窗口还不是可视的，ShowHide 就使窗口变成可视；若窗口已经是可视的，ShowHide 就不起任何作用。如果 ShowFlag 的值是 FALSE，若窗口还没处于不可视状态 ShowHide 就使窗口变成不可视，若窗口已经是不可视的 ShowHide 就不起任何作用。

ShowHide 过程并不改变窗口的醒目显示或前后次序，也不生成任何激活事件。

▲ **警告**

使用这个过程一定要小心。当您需要 HideWindow 和 ShowWindow 不能提供的其它控制部件时，您才能使用这个过程。例如，不要用 ShowHide 来隐藏当前活动的窗口，却不使另一个窗口变成前台窗口。

HiliteWindow

调用 HiliteWindow 过程设定窗口的醒目显示状态。

```
PROCEDURE HiliteWindow(theWindow:WindowPtr;fHilite:Boolear);
```

`theWindow` 窗口的窗口记录指针。

`fHilite` 确定醒目显示状态的布尔值: TRUE 醒目显示窗口; FALSE 取消醒目显示。

说明

HiliteWindow 过程将窗口的醒目显示状态设定成指定的状态。如果 fHilite 参数的值是 TRUE，HiliteWindow 就醒目显示指定的窗口；若指定的窗口已经被醒目显示，HiliteWindow 过程就不起任何作用。如果 fHilite 的值是 FALSE，HiliteWindow 就取消指定窗口的醒目显示;若该窗口已经不处于醒目显示状态，HiliteWindow 过程就不起任何作用。

您的应用程序一般并不需要调用 HiliteWindow。若要使一个窗口切换到前台来，您可调用 SelectWindow 过程，该过程将为您处理窗口的醒目显示。

BringToFront

调用 BringToFront 过程把一个窗口送到前面。

```
PROCEDURE BringToFront(theWindow:WindowPtr);
```

`theWindow` 窗口的窗口记录指针。

说明

BringToFront过程把指定的窗口放到窗口列表的表头，并在屏幕上所有其它窗口的前面重新绘制这个窗口。它并不改变窗口的醒目显示，也不使窗口变成当前活动窗口。

您的应用程序一般不调用BringToFront。用户界面准则规定最前面的窗口应该是当前活动窗口。若要把一个窗口送到前面并使它变成当前活动窗口，请调用 SelectWindow 过程。

SendBehind

调用 SendBehind 过程把一个窗口移至另一窗口的后面。

```
PROCEDURE SendBehind (theWindow, behindWindow:WindowPtr);
```

theWindow　　被移动窗口的指针。

behindWindow　　处在被移动窗口前面的窗口的指针。

说明

SendBehind 过程把参数 theWindow 指定的窗口移到参数 behindWindow 指定的窗口之后。如果这个移动使以前被遮住的窗口或窗口的某些部分暴露出来,SendBehind 就根据情况重新绘制框架并生成相应的更新事件刷新最新暴露的内容区域。

如果 behindWindow 的值是 NIL，SendBehind 就把指定的窗口移到桌面上其它所有窗口的后面。如果被移动的窗口是当前活动窗口，SendBehind 就取消它的醒目显示，醒目显示最新暴露的最前面的窗口，并生成相应的激活事件。

附　注

在您用 SelectWindow 过程使一个新窗口变成当前窗口之后，不要使用 SendBehind 来停用窗口。SelectWindow 过程自动地停用以前的前台窗口。◆

检索窗口信息

本节介绍

- FindWindow 函数，它把一个鼠标按下事件的光标位置映射到屏幕的某些部分或一个窗口的某些区域
- FrontWindow 函数，获取具体的当前活动窗口

FindWindow

当您的应用程序接收一个鼠标按下事件时，调用 FindWindow 函数将光标的位置映射到屏幕的某部分或一个窗口的某区域。

```
FUNCTION FindWindow (thePoint:Point
                     VAR theWindow:WindowPtr):Integer;
```

thePoint　在全局坐标中的点，是鼠标按下事件发生的地方。您的应用程序从事件记录的 where 域中获取此信息。

theWindow　如果鼠标按下事件发生在一个窗口中，FindWindow 就通过参数 theWindow 返回那个窗口的指针。如果鼠标按下事件没发生在窗口中，FindWindow 就把 theWindow 设定成 NIL。

说明

当用户按下鼠标按钮时，FindWindow 函数返回一个整数指定光标所处的位置。每当您接收到一个鼠标按下事件时，您都需调用 FindWindow。FindWindow 函数帮助您调度事件，当鼠标按钮被按下时它报告光标是在菜单栏中还是在一个窗口中，如果光标是在一个窗口中它还报告是哪个窗口以及窗口的区域。如果鼠标按下事件发生在一个窗口中，FindWindow 就在参数 theWindow 中放置窗口的指针。

FindWindow 函数返回一个整数，指定 9 个区域中的一个:

```
CONST  inDesk         = 0; {none of the foolowing}
       inMenuBar      = 1; {in menu bar}
       inSysWindow    = 2; {in desk accessory window}
       inContent      = 3; {anywhere in content region except size}
                           { box if window is active, }
                           { anywhere including size box if window }
                           { is inactive}
       inDrag         = 4; {in drag (title bar) region}
       inGrow         = 5; {in size box (active window only)}
       inGoAway       = 6; {in colse box}
       inZoomIn       = 7; {in zoom box (window in standard state)}
       inZoomOut      = 8; {in zoom box (window in user state)}
```

如果光标不在菜单栏、桌面附件窗口或任何属于您的应用程序的窗口中，FindWindow 就返回 inDesk。例如，如果光标在窗口框架上（但不在标题栏、关闭格或缩放格中）用户按下鼠标按钮，FindWindow 就会返回这个值。当 FindWindow 返回 inDesk 时，您的应用程序不必做任何事情。在 System 7 中，当光标在桌面上或在属于另一个应用程序的窗口中用户按下鼠标按钮时，事件管理器向您的应用程序发送一个挂起事件并切换到查找程序或另一个应用程序。

若光标在菜单栏中用户按下鼠标按钮，FindWindow 函数就返回 InMenuBar。您的应用程序一般先调整它的菜单，然后调用菜单管理器函数MenuSelect让用户选择菜单命令项目。

如果光标处在您的应用程序分区中打开的桌面附件的窗口内，此时用户按下鼠标按钮，FindWindow 函数就返回 inSysWindow。在System7 中这种情况很少出现。当用户在一个独立打开的桌面附件的窗口中点按鼠标时，事件管理器向您的应用程序发送一个挂起事件并切换到桌面附件。

如果 FindWindow 返回了 inSysWindow，您的应用程序就调用 SystemClick 过程，在本书的"事件管理器"一章中有关于这个过程的说明。SystemClick 过程把事件送到桌面附件。如果用户在光标处于跑在后台桌面附件窗口的内容区域时按下鼠标按钮，SystemClick就向您的应用程序和桌面附件分别发送相应的激活事件，使桌面附件窗口变成当前活动窗口。

如果光标处于您的应用程序的一个窗口的内容区（不包括后台窗口中的尺寸格），此时用户按下鼠标按钮，Findwindow 函数就返回 inContent。您的应用程序需调用自己的例程处理内容区中的鼠标单击。

如果光标在一个窗口拖拽区（即标题栏，不包括关闭格和缩放格）中用户按下鼠标按钮，FindWindow 函数就返回 inDrag。您的应用程序随后调用窗口管理器的 DragWindow 过程让用户把窗口拖拽到一个新位置。

如果光标在当前活动窗口的尺寸格中用户按下鼠标按钮，FindWindow 函数就返回 inGrow。您的应用程序就调用它自己的例程调整窗口大小。

如果光标在当前活动窗口的关闭格中用户按下鼠标按钮，FindWindow 函数就返回 inGoAway。在鼠标被按下时您的应用程序调用 TrackGoAway 函数跟踪鼠标活动，如果光标在关闭格中用户释放鼠标按钮，您的应用程序就调用它自己的例程来关闭窗口。

如果光标在当前活动窗口的缩放格中用户按下鼠标按钮，FindWindow 函数就返回 inZoomIn 或 inZoomOut。在鼠标被按下时您的应用程序调用 TrackBox 函数跟踪鼠标活动，如果用户释放鼠标按钮光标仍在缩放格中时，您的应用程序就调用它自己的例程来缩放窗口。

亦请参阅

请参阅4-44页程序清单4-9中关于调用FindWindow来确定光标位置并根据所得结果调度鼠标事件的实例。

FrontWindow

调用 FrontWindow 找出哪个窗口是当前活动窗口。

```
FUNCTION FrontWindow:WindowPtr;
```

说明

FrontWindow 函数返回窗口列表中第一个可视窗口（即当前活动窗口）的指针。如果没有任何可视窗口，FrontWindow 就返回 NIL。

亦请参阅

请参阅 4-44 页程序清单 4-9 中关于调用 FrontWindow 确定一个事件是否发生在当前活动窗口中的实例。

请参阅 4-55 页程序清单 4-12 中关于调用 FrontWindow 确定在改变窗口大小后是否在其它窗口前面显示该窗口的实例。

请参阅 4-60 页的程序清单 4-16 和 4-61 页的程序清单 4-17 中的实例，它们调用 FrontWindow- 确定用户操作命令直接影响的当前活动窗口的指针。

移动窗口

本节介绍在桌面上移动窗口的过程。

若要移动一个窗口，您的应用程序一般仅需调用 DragWindow 过程（这个过程它自身又调用 DragGrayRgn 函数）和 MoveWindow 过程。只要用户按住鼠标按钮（此时鼠标位置处于窗口拖拽区），DragGrayDgn 函数就在屏幕上随着光标的运动拖拽以虚线表示的窗口轮廓。DragGrayRgn 函数它自身调用 PinRect 函数来包含鼠标按钮在可用的桌面区域内第一次被按下时光标所在的点。当用户释放鼠标按钮时，DragWindow 调用 MoveWindow，MoveWIndow 把窗口移动到一个新位置。

DragWindow

当用户通过拖拽窗口的标题栏拖拽窗口时，使用 DragWindow 过程在屏幕上移动窗口。

```
PROCEDURE   DragWindow (theWindow:WindowPtr;
                              startPt:Point; boundsRect:Rect);
```

theWindow 被拖拽窗口的窗口记录指针。

startPt 用户按下鼠标按钮时光标在全局坐标中的位置。您的应用程序从事件记录的 where 域中获取该信息。

boundsRect　一个以全局坐标标记的矩形，它限制了窗口可以拖拽的区域。若光标释放时鼠标按钮在 boundsRect 限制的区域之外，DragWindow 不移动窗口就返回　或者，对于后台窗口，不把它切换到后台来)。

由于用户通常不能将光标移出桌面，所以当您使用 DragWindow 跟踪移动鼠标时，您可以安全地把 boundRect 设定为最大的可用矩形　全局变量 GrayRgn 指向的桌面区域的约束方框)。不要把约束矩形设定为屏幕的大小　screenBits. bounds)，因为这将使用户不能把窗口移至多显示器系统的其它屏幕上。

说明

DragWindow 过程按照光标的移动，在屏幕上移动指定窗口的虚线轮廓线，直至用户释放鼠标按钮。当按钮释放后，DragWindow 调用 MoveWindow 把窗口移至新的位置。如果指定的窗口不是一个当前活动窗口　而且当鼠标按钮按下时，Command 键并未按下)，DragWindow 就通过调用 MoveWindow 把 front 参数值设定为 TRUE，使它成为当前活动窗口。如果鼠标按钮按下时，同时也按下了 Command 键，DragWindow 仅移动窗口，但不使它成为当前活动窗口。

亦请参阅

DragWindow 过程既调用 MoveWindow，也调用 DragGrayRgn，它们将在本节中说明。

如果用户按下鼠标按钮的同时，光标正在拖拽区域中，这种情况下调用 DragWindow 的实例，请参阅 4-44 页的程序清单 4-9。

MoveWindow

使用 MoveWindow 过程在桌面环境中移动窗口。

```
PROCEDURE MoveWindow  (theWindow :  WindowPtr ;
                       hGlobal,  vGlobal :  Integer ;
                       front :  Boolean) ;
```

theWindow　一个被移动窗口的窗口记录的指针。

hGlobal　窗口端口矩形的左边界在全局坐标中的新位置。

vGlobal　窗口端口矩形的上边界在全局坐标中的新位置。

`front` 一个指定窗口是否变成最前面的、当前活动窗口的布尔值。如果 front 参数的值是FALSE，MoveWindow就不更改它所在的平面或状态。如果 front 参数的值是 TRUE，而且窗口不是当前活动窗口，MoveWindow 就通过调用 SelectWindow 过程将其切换到前台来。

说明

MoveWindow 过程把指定的窗口移至 hGlobal 和 vGlobal 参数指定的位置，但不更改它的大小。窗口端口矩形的左上角搁放在点（vGlobal，hGlobal）上。左上角的局部坐标不受影响。

通常您的应用程序不调用 MoveWindow。当用户通过拖拽窗口的标题栏拖拽窗口时，您可以调用 DragWindow，当用户释放鼠标按钮时，再由 DragWindow 去调 MoveWindow。

DragGrayRgn

当用户拖拽窗口时，DragWindow 函数调用 DragGrayRgn 在屏幕上移动窗口的轮廓线。

```
FUNCTION DragGrayRgn (theRgn: RgnHandle; startPt: Point;
                      limitRect, slopRect: Rect; axis: Integer;
                      actionProc: ProcPtr): LongInt;
```

`theRgn` 被拖拽区域的一个间接指针。

`startPt` 当按下鼠标按钮时，光标在当前图形端口的局部坐标中的位置。

`limitRect` 在当前图形端口的局部坐标中限制拖拽区域的矩形。该参数和 slopeRect 参数一起使用，请参阅 4-98 页中的图 4-23。

`slopeRect` 在当前图形端口的局部坐标中的矩形，它在不违背 limitRect 参数的限制下，能够给用户移动窗口留出一些余地，请参阅 4-98 页中的图 4-23。slopeRect 矩形应该比 limitRect 矩形大一些。

`axis` 约束区域移动的常数。axis 参数可以从以下所列值获取:

```
CONST noConstraint  = 0;    {no constraints}
      hAxisOnly     = 1;    {move on horizontal axis }
                            { only}
      vAxisOnly     = 2;    {move on vertical axis }
                            { only}
```

如果轴约束有效，轮廓线只沿指定的轴跟随光标移动，而忽略沿其它轴的移动。无论是否有轴约束，只有当鼠标在 slopeRect 矩形内时，轮廓线才出现。

actionProc 一个过程的指针，该过程定义了一个动作，只要用户按下鼠标，就须重复执行这个动作。该过程可以没有参数。若 actionProc 的值是 NIL，DragGrayRgn 简单地保留控制，直至释放鼠标按钮。

说明

DragGrayRgn 函数按照光标的移动，移动屏幕上一个区域的灰色轮廓线，直至释放鼠标按钮。它返回的是鼠标按钮按下的点和偏移点的差，偏移点对该区域包容矩形左上角的水平和垂直偏移等于用户按下鼠标按钮的起始点的偏移。DragGrayRgn 函数在返回值的高位字储存起始点和偏移点的垂直差，在低位字储存水平差。

DragGrayRgn 函数按照 limitRect 和 slopeRect 设定的约束，限制区域的移动：

- 只要光标在 limitRect 矩形内，区域的轮廓线就正常地跟随光标移动。如果释放鼠标的同时，光标在该矩形内，返回的值反映了光标在每个方向移动的距离
- 当光标移出了 limitRect 矩形时，偏移点停在 limitRect 矩形的边上。如果释放鼠标的同时，光标在 limitRect 矩形外但在 slopeRect 矩形内，返回的值只反映开始点和偏移点的差，而不管光标已经移出 limitRect 矩形多远（注意，区域的那个部分可以落到 limitRect 区域之外，但是偏移点不能）
- 当光标移出 slopeRect 矩形时，区域的轮廓线在屏幕上消失。但是，DragGrayRgn 函数继续跟踪光标，如果光标移回 slopeRect 矩形，轮廓线就重新出现。如果释放鼠标的同时，光标在 slopeRect 矩形外，返回值的高位字和低位字都设定为 $ 8000。这时，窗口管理器保持窗口原来的位置不变

4-98 页的图 4-23 说明了当偏移点到达 limitRect 矩形的边时，区域如何停止移动。这时，即使光标继续移动，区域也不移动。

如果释放鼠标的同时，光标在 slopeRect 矩形内的任何地方，窗口管理器在新位置重新绘制窗口，这个位置是由 DragGrayRgn 返回的值计算出的。



图 4-23　DragGrayRgn 使用的限制矩形

汇编语言信息

您可以把全局变量 DragHook 设定为指向您的应用程序的一个可选过程，只要按下了鼠标按钮，DragGrayRgn 将调用这个可选过程。(如果有一个 actionProc 过程，它将先被调用。) 如果您要 DragGrayRgn 绘制区域的轮廓线，但其图案不是灰色，您可以在全局变量 DragPattern 中储存该图案，然后请求调用 macro _DragTheRgn。注意，不能保证窗口管理器全局变量的使用和 System 6 以后的系统软件版本兼容。

PinRect

DragGrayRgn 函数使用 PinRect 函数包含在一个指定矩形中的点。

```
FUNCTION PinRect (theRect: Rect; thePt: Point): LongInt;
```

theRect　　被包含的点所在的矩形。

thePt　　被包含的点。

说明

PinRect 函数返回在指定矩形中的一个点，它和指定的点尽可能接近。(返回的长整数的高位字是垂直坐标，低位字是水平坐标。)

如果指定的点在矩形中，PinRect 就返回这个点。如果不在:

- 如果水平位置在矩形的左面，PinRect 返回矩形左边作为水平坐标
- 如果水平位置在矩形的右面，PinRect 返回矩形右边减 1 作为水平坐标
- 如果垂直位置在矩形的上面，PinRect 返回矩形顶边作为垂直坐标
- 如果垂直位置在矩形的下面，PinRect 返回矩形底边减 1 作为垂直坐标

附　注:
当点在矩形的下面和右面时，要减去 1，使得在这个点绘制的象素在矩形内。但是，如果这个点恰好在矩形的底边或右边，虽应该减去 1 但是没有减。

调整窗口大小

本节说明一个过程，当用户调整窗口大小时，您可以用它跟踪光标，并按新的尺寸绘制窗口。

GrowWindow

应使用GrowWindow函数允许用户更改窗口的大小。当用户移动光标放大或缩小窗口时，GrowWindow 函数显示窗口的轮廓线；它处理和用户所有的对话，直至用户释放鼠标按钮。在调用了 GrowWindow 之后，您再调用 SizeWindow 过程更改窗口的大小。

```
FUNCTION GrowWindow (theWindow: WindowPtr;
                     startPt: Point; sizeRect: Rect): LongInt;
```

`theWindow` 被拖拽的窗口的窗口记录间接指针。

`startPt` 鼠标按钮首次按下时光标所在位置，用全局坐标表示。您的应用程序从事件记录的 where 域检索这个点的坐标值。

`sizeRect` 端口矩形垂直和水平测量的极限，以像素为单位。

虽然 sizeRect 参数是 Rect 数据类型的格式，其结构中的 4 个数表示长度，不是屏幕坐标。sizeRect 参数的 top、left、bottom 和 right 域指定了最小的垂直高度（top）、最小的水平长度（left）、最大的垂直高度（bottom）以及最大的水平长度（right）。

最小长度必须足够大，以便于矩形的管理；每边 64 个像素是一个典型值。因为通常用户不能把光标移出屏幕，当您使用 GrowWindow 跟踪光标移动时，您可以安全地把最大可能长度（65,535 个像素）设定为上限。

说明

GrowWindow 函数按照光标的移动，在屏幕上移动窗口的右边和底边的虚线图像，直至释放鼠标按钮。它返回由此得到的窗口的尺寸，以像素为单位：返回长整数的高位字是高度，低位字是宽度。您可以使用 HiWord 和 LoWord 分别检索高位字和低位字。

若返回值为 0，意味着新的大小和当前端口矩形的大小相同。

汇编语言信息

您可以把全局变量DragHook设定为指向一个可选过程，它由您的应用程序定义，只要按下了鼠标按钮，GrowWindow 将调用这个可选过程。（如果有一个 actionProc 过程，它将先被调用。）注意，不能保证窗口管理器全局变量的使用和 System 6 以后的系统软件版本兼容。

亦请参阅

如果用户按下鼠标按钮的同时，光标正处在尺寸格中，这种情况下调用 GrowWindow 的实例，请参阅 4-44 页的程序清单 4-9。

SizeWindow

应使用 SizeWindow 设定窗口的大小。

```
PROCEDURE SizeWindow (theWindow: WindowPtr; w, h: Integer;fUpdate: Boolean);
```

theWindow 调整大小的窗口的窗口记录间接指针。

w 窗口新宽度，以像素为单位。

h 窗口新高度，以像素为单位。

fUpdate 一个布尔值，它指定内容区域新创建的一块面积是否应该累积在更新区域中。如果应该，它是 TRUE，如果不应该，它是 FALSE。通常您传递 TRUE 值，以便保证对该面积进行更新。如果您传递的是 FALSE，您应该负责自己保留更新区域。有关向更新区域添加或从更新区域删除矩形的信息，请参阅 4-107 页对 InvalRect 的说明及 4-108 页对 ValidRect 的说明。

说明

SizeWindow 过程把窗口的图形端口矩形的大小更改成由 w 和 h 参数指定的尺寸。如果 w 和 h 的值是 0，它不作任何事情。窗口管理器按新的大小重新绘制窗口，重新把标题居中，如果必要，就截断它。在调用 GrowWindow 之后，您的应用程序应立即调用 SizeWindow，以便按照用户通过尺寸格所做的更改调整窗口。

亦请参阅

有关调用 SizeWindow，按照 GrowWindow 返回的值调整窗口大小的实例，请参阅 4-58 页的程序清单 4-13。

缩放窗口

您可以使用本节说明的过程跟踪缩放格中的鼠标活动，并缩放窗口。

TrackBox

如果用户按下鼠标按钮的同时，光标正处在缩放格中，应使用 TrackBox 函数跟踪光标的移动。

```
FUNCTION TrackBox (theWindow: WindowPtr; thePt: Point;partCode: Integer): Boolean;
```

theWindow 窗口的窗口记录指针，鼠标按钮在该窗口中按下。

thePt 按下鼠标按钮时的光标所在位置。您的应用程序可从事件记录的where域中收到获取该位置信息。

partCode FindWindow 函数返回的部分代码（inZoomIn 或 inZoomOut）。

说明

如果用户按下鼠标按钮的同时，光标处在缩放格中，TrackBox 函数就跟踪光标，在释放鼠标按钮之前，它保持控制。正如 4-47 页中的图 4-20 所说明的那样，鼠标按钮按下时，只要光标在缩放区域，TrackBox 就醒目显示缩放格。

当释放鼠标按钮时，TrackBox 取消缩放格的醒目显示，如果光标在缩放区域，它返回 TRUE，如果不在，它返回 FALSE。

当您的应用程序从 FindWindow 函数得到结果代码 inZoomIn 或 inZoomOut 时，它就调用 TrackBox 函数。如果 TrackBox 返回 TRUE，您的应用程序对标准状态进行必要的计算，并调用 ZoomWindow 过程缩放窗口。如果 TrackBox 返回 FALSE，您的应用程序不做任何事情。

汇编语言信息

您可以把全局变量 DragHook 设定为指向一个可选过程，它由您的应用程序定义，只要按下了鼠标按钮，TrackBox 将调用这个可选过程。(如果有一个 actionProc 过程，它将先被调用。) 注意，不能保证窗口管理器全局变量的使用和 System 6 以后的系统软件版本兼容。

亦请参阅

如果用户按下鼠标按钮的同时，光标正在缩放格中，这种情况下调用 TrackBox 跟踪光标活动的实例，请参阅 4-55 页的程序清单 4-12

ZoomWindow

如果用户按下及释放鼠标按钮的同时，光标均处在在缩放格中，应使用 ZoomWindow 过程缩放窗口。

```
PROCEDURE ZoomWindow (theWindow: WindowPtr;
                      partCode: Integer; front: Boolean);
```

theWindow 缩放窗口的窗口记录指针。

partCode FindWindow 函数返回的结果（inZoomIn 或 inZoomOut）。

front 一个布尔值，它决定该窗口是否带到前台。如果 front 的值是 TRUE，窗口必定变成最前面的当前活动窗口。如果 front 的值是 FALSE，该窗口在窗口列表中的位置不变。注意，如果窗口在缩放前就是激活的，即使 front 的值是 FALSE，它仍然保持激活状态。

说明

ZoomWindow 过程究竟是放大还是缩小窗口，取决于 partCode 参数的值。当您的应用程序从 TrackBox 收到 TRUE 时，它调用 ZoomWindow，把 FindWindow 返回的部分代码传送给 ZoomWindow。然后，ZoomWindow 过程把窗口的端口矩形更改成用户状态 如果 partCode 参数值码是 inZoomIn）或者标准状态 如 partCode 参数值是 inZoomOut），这两个状态储存在窗口状态数据记录中。这些情况的说明在 4-53 页的“缩放窗口”一节中。

亦请参阅

有关计算和设定标准状态，然后调用 ZoomWindow 缩放一个窗口的实例，请参阅 4-55 页的程序清单 4-12。

关闭窗口并取消分配给窗口的存储器

下一段文字描述了在关闭框中用户的动作，关闭窗口并释放窗口的过程。当您不再需要一个窗口的时候，如果你为窗口申请了内存，请调用 CloseWindow 过程；否则请调用 DisposeWindow 过程。

TrackGoAway

如果用户按下鼠标按钮的同时，光标在关闭格中，应使用 TrackGoAway 函数跟踪光标。

```
FUNCTION TrackGoAway  (theWindow : WindowPtr ;
                        thePt : Point) : Boolean ;
```

theWindow 窗口的窗口记录的一个指针，鼠标按下事件发生在该窗口中。

thePt 鼠标按钮按下时光标的位置。您的应用程序从事件记录的 where 域中获取该位置信息

说明

如果用户按下鼠标按钮的同时，光标在关闭格中，TrackGoAway跟踪光标的活动，它保持控制，直至用户释放鼠标按钮。4-46页的图4-19 说明了只要光标在关闭区域，当鼠标按钮按下时，TrackGoAway 就醒目显示关闭格。

当释放鼠标按钮时，TrackGoAway取消关闭格的醒目显示，如果光标在关闭格内，就返回 TRUE，如果在关闭格外，就返回 FALSE。

当您的应用程序从 FindWindow 收到一个结果代码 inGoAway 时，就调用 TrackGoAway 函数。如果 TrackGoAway 返回 TRUE，您的应用程序调用它自己的过程关闭窗口，这个过程可以调用 CloseWindow 过程或 DisposeWindow 过程从屏幕上取消窗口。[在取消文稿窗口前，您的应用程序通常要进行检查，检查上次存储的文件是否做过修改。有关处理文件的一般讨论，请参阅《Inside Macintosh 文件》(*Inside Macintosh: Files*) 一书。] 如果 TrackGoAway 返回 False，您的应用程序不做任何事情。

汇编语言信息

您可以把全局变量 DragHook 设定为指向一个可选过程，它由您的应用程序定义，只要按下了鼠标按钮，TrackGoAway将调用这个可选过程。(如果有一个 actionProc 过程，它将先被调用。) 注意，不能保证窗口管理器全局变量的使用和 System 6 以后的系统软件版本兼容。

亦请参阅

如果用户按下鼠标按钮的同时，光标正在关闭格中，这种情况下调用 TrackGoAway 跟踪光标活动的实例，请参阅 4-44 页的程序清单 4-9。

CloseWindow

如果您自己为窗口的窗口记录分配了内存空间，应使用CloseWindow过程取消窗口。

```
PROCEDURE CloseWindow (theWindow: WindowPtr);
```

theWindow　　窗口的窗口记录的一个指针，该窗口要被关闭。

说明

CloseWindow过程从屏幕上取消指定的窗口，并从窗口列表中删除它。该过程释放除窗口记录外，窗口的所有数据结构占据的内存空间。

如果窗口记录的内存空间是由您分配的，并给它传回了一个指针作为创建窗口的函数的参数，当您完成了该窗口的任务后，应调用 CloseWindow。然后您必须调用内存管理器的过程 DisposePtr 释放窗口记录占据的内存空间。

警告

如果您的应用程序分配了其它的内存空间和窗口一起使用，您在调用CloseWindow之前，必须先释放这些内存空间。窗口管理器只释放它所创建的数据结构。

CloseWindow还假定窗口记录域windowPic指向的图片是数据，而不是资源，它调用 QuickDraw 的过程 KillPicture 删除图片。如果您的应用程序使用一个作为资源储存的图片，您必须用 ReleaseResource 过程释放它占据的存储器，并在关闭窗口前，把 windowPic 域设定为 NIL。

该窗口所有等待处理的更新事件都被删除。如果要取消的窗口是最前面的的窗口，在它后面的窗口（如果有的话）就变成当前活动窗口。

亦请参阅

有关调用CloseWindow从屏幕上取消窗口的实例，请参阅4-61 页的程序清单 4-17。

当试图创建窗口失败时，调用 CloseWindow 清除内存空间的实例，请参阅 4-28 页的列表 4-3。

DisposeWindow

如果窗口管理器为窗口记录分配内存空间，应使用 DragWindow 过程取消窗口。

```
PROCEDURE DisposeWindow (theWindow: WindowPtr);
```

theWindow　　窗口的窗口记录的一个指针，该窗口要被关闭。

说明

DisposeWindow 过程从屏幕上取消指定的窗口，并从窗口列表中删除它。该过程释放窗口的所有数据结构占据的内存空间，包括窗口记录。（DisposeWindow 调用 CloseWindow，然后释放被窗口记录占据的内存空间。）

警告

如果您的应用程序分配了其它的内存空间和窗口一起使用，您在调用 DisposeWindow 之前，必须先释放这些内存空间。窗口管理器只释放它创建的数据结构。

DisposeWindow 假定窗口记录域 windowPic 指向的图片是数据，而不是资源，它调用 QuickDraw 的过程 KillPicture 删除图片。如果您的应用程序使用一个作为资源储存的图片，您必须用 ReleaseResource 过程释放它占据的存储器，并在关闭窗口前，把 windowPic 域设定为 NIL。

该窗口所有等待处理的更新事件都被删除。如果要取消的窗口是最前面的的窗口，在它后面的窗口（如果有的话）就变成当前活动窗口。

保留更新区域

本节说明您用来更新窗口及保留窗口更新区域的例程。

BeginUpdate

当您收到一个窗口的更新事件时，应使用 BeginUpdate 过程开始更新该窗口。

```
PROCEDURE BeginUpdate (theWindow: WindowPtr);
```

theWindow　　窗口的窗口记录的一个指针。您的应用程序从更新事件记录的 message 域中获得这个信息。

说明

BeginUpdate过程首先把窗口图形端口的可视区域限制为可视区域和更新区域的重叠部分；然后将该窗口的更新区域置空。您的应用程序在调用 BeginUpdate 后，重新绘制整个内容区域或只绘制可视区域。在这两种情况下，屏幕上真正重新绘制的只是窗口中要求更新的那部分。

每次调用 BeginUpdate 后，必须调用 EndUpdate 一次，EndUpdate 应在您的应用程序重新绘制内容区域后调用。

附注：

BeginUpdate 和 EndUpdate 在 Pascal 中不能嵌套。也就是说，您必须在下一次调用 BeginUpdate 之前，调用 EndUpdate。

如果您存储并还原 visRgn 的一个备份，BeginUpdate 和 EndUpdate 的调用，在汇编语言中可以嵌套，

visRgn 的备份以全局坐标储存在全局变量 SaveVisRgn 中。

特殊考虑

如果您收到一个更新事件时，应先清除其更新区域，否则事件管理器将继续向您发送更新事件，直至将更新区域置空。

亦请参阅

有关 BeginUpdate 和 EndUpdate 如何影响可视区域和更新区域的说明，请参阅 4-49 页的图 4-21。有关更新一个窗口的实例，请参阅 4-50 页的程序清单 4-10.

EndUpdate

应使用 EndUpdate 过程完成窗口的更新。

```
PROCEDURE EndUpdate (theWindow: WindowPtr);
```

theWindow　　窗口的窗口记录的一个指针。

说明

EndUpdate 过程还原窗口图形端口的正常可视区域。当您收到一个窗口的更新事件时，您调用 BeginUpdate，重新绘制更新区域，然后调用 EndUpdate。每次调用 BeginUpdate 以后，必须调用 EndUpdate 一次。

亦请参阅

有关 BeginUpdate 和 EndUpdate 如何影响可视区域和更新区域的说明，请参阅 4-49 页的图 4-21。有关更新一个窗口的实例，请参阅 4-50 页的程序清单 4-10.

InvalRect

应使用 InvalRect 把一个矩形区域添加至窗口的更新区域。

badRect　一个以局部坐标表示的矩形，它将被添加至窗口的更新区域。

说明

InvalRect过程把一个指定的矩形添加至窗口的更新区域，该窗口的图形端口是当前端口。应在局部坐标系中指定该矩形。为了适合窗口内容区，如果必要，窗口管理器将对它进行剪辑。

您的应用程序和窗口管理器都使用InvalRect过程。例如，当用户放大一个窗口时，窗口管理器使用InvalRect把新创建的内容区域添加至更新区域。您的应用程序使用

InvalRgn

应使用 InvalRgn 把一个区域添加至窗口的更新区域。

```
PROCEDURE InvalRgn (badRgn: RgnHandle);
```

badRect　一个以局部坐标表示的矩形，它将被添加至窗口的更新区域。把原来被滚动杆占据的两个矩形添加至原来的内容区域。

说明

InvalRgn 过程把一个指定的区域添加至窗口的更新区域，该窗口的图形端口是当前端口。应在局部坐标系中指定该区域。为了适合窗口内容区，如果必要，窗口管理器将对它进行剪辑。

亦请参阅

有关使用InvalRgn把窗口内容区域的一部分添加至更新区域的实例，请参阅 4-58 页的程序清单 4-13。

ValidRect

应使用 ValidRect 过程从窗口的更新区域中删除一个矩形。

```
PROCEDURE ValidRect (goodRect: Rect);
```

goodRect　一个以局部坐标表示的矩形，它将从窗口的更新区域中删除。

说明

ValidRect过程从窗口的更新区域中删除一个指定的矩形,该窗口的图形端口是当前端口。应在局部坐标系中指定该矩形。为了适合窗口的内容区域，如果必要，窗口管理器将对它进行剪辑。

您的应用程序使用 ValidRect 通知窗口管理器，它已经绘制了一个矩形，它还使用 ValidRect 取消所有积累的对该区域的更新要求。从而，您可以通过减少多余的重新绘制任务，提高响应。

例如，假定您已经调整了一个窗口的大小，该窗口包含一个尺寸格和滚动杆。新的尺寸格和滚动杆可能在，也可能不在窗口的更新区域内，这取决于调整了大小的新窗口的尺寸。在调用 SizeWindow 之后，您可以立即重新绘制尺寸格和滚动杆，然后对它们占据的面积调用 ValidRect。如果它们已经被累加至更新区域，ValidRect 就删除它们，这样，在下次更新事件中您就不需要重新绘制它们。

亦请参阅

有关使用ValidRect从更新区域中删除窗口内容区域一部分的实例，请参阅 4-58 页中的程序清单 4-13。

ValidRgn

应使用 ValidRgn 从窗口的更新区域中删除一个指定的区域。

```
PROCEDURE ValidRgn (goodRgn: RgnHandle);
```

goodRect 一个以局部坐标表示的区域，它将从窗口的更新区域中删除。

说明

ValidRgn 过程从窗口的更新区域中删除一个指定的区域，该窗口的图形端口是当前端口。应在局部坐标系中指定该区域。为了适合窗口的内容区域，如果必要，窗口管理器将对它进行剪辑。

设定和检索其它窗口特征

本节介绍的例程，允许您在窗口记录中设定及检索不常用的域。

SetWindowPic

应使用SetWindowPic过程建立一个图片，窗口管理器可以在窗口的内容区域绘制该图片。

```
PROCEDURE SetWindowPic (theWindow: WindowPtr;Pic: PicHandle);
```

theWindow 窗口的窗口记录的一个指针。

Pic 要在窗口中绘制的图片的间接指针。

说明

SetWindowPic过程在窗口的窗口记录中储存一个图片的间接指针，该图片将要在窗口中绘制。当窗口的内容区域必须被更新时，如果必要，窗口管理器绘制该图片或它的一部分，而不生成一个更新事件。

附注:

CloseWindow和DisposeWindow过程假定窗口记录的域windowPic 指向的所有图片都是作为数据，而不是作为资源储存的。如果您的应用程序使用一个作为资源储存的图片，您必须在关闭窗口前，调用资源管理器的ReleaseResource过程释放该图片占据的内存空间，并把windowPic域设定为 NIL。

GetWindowPic

应使用GetWindowPic函数为窗口图片检索的一个间接指针。

```
FUNCTION GetWindowPic (theWindow: WindowPtr): PicHandle;
```

theWindow 窗口的窗口记录的一个指针。

说明

GetWindowPic函数给要在指定的窗口内容区中绘制的图片返回一个间接指针。这个二级指针必须是原来用 SetWindowPic 过程储存的。

SetWRefCon

应使用 SetWRefCon 过程设定窗口记录的 refCon 域。

```
PROCEDURE SetWRefCon (theWindow: WindowPtr; data: LongInt);
```

theWindow　　窗口的窗口记录的一个指针。

data　　要放置在 refCon 域中的数据。

说明

SetWRefCon 过程在指定窗口记录的 refCon 域中放置指定的数据。您的应用程序可以使用 refCon 文件储存任何有关窗口的数据。

亦请参阅

有关设定 refCon 域的实例请参阅 4-28 页的程序清单 4-3。有关使用 refCon 域内容的实例请参阅 4-60 页的程序清单 4-16。

GetWRefCon

应使用 GetWRefCon 函数从窗口的窗口记录中检索参考常数。

```
FUNCTION GetWRefCon (theWindow: WindowPtr): LongInt;
```

theWindow　　窗口的窗口记录的一个指针。

说明

GetWRefCon 函数返回储存在指定窗口记录的 refCon 域中的长整型数据。

亦请参阅

有关建议使用 refCon 文件的方式，请参阅 4-23 页的“管理多个窗口”一节。有关获取 refCon 域的应用程序定义例程的实例，请参阅 4-25 页的程序清单 4-1。

GetWVariant

应使用 GetWVariant 函数检索窗口的变量代码。

```
FUNCTION GetWVariant (theWindow: WindowPtr): Integer;
```

theWindow　　窗口的窗口记录的一个指针。

说明

GetWVariant 函数返回指定窗口的变量代码。GetWVariant 的结果可以代表标准窗口类型（在 4-25 页的“创建一个窗口”一节中列出）之一，或者您自己的窗口定义函数定义的变量代码，何种结果取决于窗口的窗口定义函数。

亦请参阅

有关变量代码的定义请参阅 4-8 页开始的“窗口类型”一节。有关变量代码的详细说明，请参阅 4-120 页的“窗口定义函数”一节。

操纵桌面环境

本节介绍的例程，允许您的应用程序检索关于桌面环境的信息，并设定桌面图案。通常您的应用程序不需要操纵桌面环境中位于它自己窗口之外的任何部分。

SetDeskCPat

应使用 SetDeskCPat 过程在支持彩色 QuickDraw 的电脑上设定桌面图案。

```
PROCEDURE SetDeskCPat (deskPixPat: PixPatHandle);
```

deskPixPat　像素图案的一个间接指针。

说明

SetDeskCPat过程把桌面图案设定为一种指定的像素图案，这种图案可以用两种以上的彩色绘制。SetDeskCPat 被调用一次以后，桌面环境自动按新的图案绘制。如果指定的图案是一个二进制图案(图案类型为 0)，被绘制的是当前的前台和后台彩色。如果 deskPixPat 参数是 NIL，SetDeskCPat 就使用标准的二进制桌面图案（即，资源标识符为 16 的‘ppat’资源)。

附注:

为了和其它Macintosh应用程序及系统软件兼容，应用程序在一般情况下不应该改变桌面图案。

窗口管理器的桌面涂色例程可以用两种方式为桌面环境涂色：储存在全局变量DeskPattern中的二进制图案，或新的像素图案。电脑启动时使用的桌面图案由参数RAM位标志的值决定，该参数称为pCDeskPat。如果pCDeskPat的值为 0，窗口管理器使用新的像素图案；如果不为 0，就使用储存在DeskPattern中的二进制图案。用户可以通过通用控制板更改彩色图案，易用控制板可以更改pCDeskPat的值。

GetGrayRgn

应使用GetGrayRgn函数为当前桌面区域检索一个间接指针。

```
FUNCTION GetGrayRgn: RgnHandle;
```

说明

GetGrayRgn函数从全局变量GrayRgn为当前桌面区域返回一个间接指针。

桌面区域是指所有可用的屏幕空间，即，附属于电脑的所有显示器显示的桌面面积。通常，您的应用程序不需要直接存取桌面区域。

当您的应用程序为使用户能够拖拽一个窗口，而调用DragWindow时，它可以使用GetGrayRgn为整个桌面面积设定一个限制矩形。

亦请参阅

为使用户能够移动窗口，需要调用DragWindow，这种情况下的一个使用GetGrayRgn指定限制矩形的实例，请参阅 4-44 页的程序清单4-9。

GetCWMgrPort

在支持彩色QuickDraw的系统上，应使用GetCWMgrPort过程检索窗口管理器端口的一个指针。

```
PROCEDURE GetCWMgrPort (VAR wMgrCPort: CGrafPtr);
```

wMgrCPort　　一个参数，GetCWMgrPort 在该参数中返回窗口管理器端口的指针。

说明

GetCWMgrPort 过程在参数wMgrCPort中放置彩色窗口管理器端口的一个指针。只能在带有彩色 QuickDraw 的电脑上使用 GetCWMgrPort 过程。

窗口管理器端口是一个图形端口，它占据了主屏幕的全部面积。通常，您的应用程序不需要存取窗口管理器端口。

附注:

不要更改窗口管理器端口的任何区域。如果您更改了它，窗口管理器可能会不正确地处理重叠的窗口。

GetWMgrPort

在只有原单色QuickDraw的系统上，应使用GetWMgrPort过程检索窗口管理器端口的一个指针。

```
PROCEDURE GetWMgrPort (VAR wPort: GrafPtr);
```

wPort　　一个参数，GetWMgrPort 在该参数中返回窗口管理器端口的指针。

说明

GetWMgrPort 过程在参数 wPort 中放置窗口管理器端口的一个指针。

窗口管理器端口是一个图形端口，它占据了主屏幕的全部面积。通常，您的应用程序不需要存取窗口管理器端口。

附注:

改窗口管理器端口的任何区域。如果您更改了它，窗口管理器可能会不正确地处理重叠的窗口。

操纵窗口彩色信息

本节介绍您用来设定及检索窗口色彩信息的例程。在正常情况下，您的应用程序不应更改窗口色彩信息。

SetWinColor

应使用 SetWinColor 过程设定窗口的窗口色彩表。

```
PROCEDURE SetWinColor (theWindow: WindowPtr;
                       newColorTable: WCTabHandle);
```

theWindow　　窗口的窗口记录的一个指针。

newColorTable 窗口彩色表的一个间接指针，它为新的窗口色彩表定义了色彩。

说明

SetWinColor 过程设定窗口的色彩表。如果窗口没有辅助的窗口记录，该过程用指定的窗口色彩表创建一个新的辅助记录，并把它添加至辅助窗口列表。如果窗口已经有一个辅助记录，将其窗口色彩表替换。然后窗口管理器重新绘制窗口框架，以新的色彩醒目显示文字，并把窗口的背景色设定为新的内容色彩。

如果新的色彩表的输入项和系统设定色彩表一样，SetWinColor 更改辅助窗口记录，使它指向系统设定色彩表。

窗口彩色表资源（ 'wctb' 类型的资源）应该是不能清除的。

如果您对 theWindow 参数指定一个值 NIL，SetWinColor 更改存储器中的系统设定色彩表。但是，您的应用程序不应该更改系统设定色彩表。

亦请参阅

有关窗口色彩表的说明，请参阅 4-71 页中的“窗口彩色表记录”一节。有关辅助窗口记录的说明，请参阅 4-73 页中的“辅助窗口记录”一节。有关 'wctb' 资源的说明，请参阅 4-127 页中的“窗口彩色表资源”一节。

GetAuxWin

应使用 GetAuxWin 函数检索窗口的辅助窗口记录间接指针。

```
FUNCTION GetAuxWin (theWindow: WindowPtr;
                    VAR awHndl: AuxWinHandle): Boolean;
```

theWindow　　窗口的窗口记录的一个指针。

awHndl　　窗口的辅助窗口记录间接指针。

说明

GetAuxWin 函数返回一个布尔值，这个值报告窗口是否有一个辅助窗口记录，GetAuxWin 还把可变参数 awHndl 设定到窗口的辅助窗口记录中。

如果窗口没有辅助窗口记录，GetAuxWin 在 awHndl 中放置系统设定窗口色彩表，并返回一个值 FALSE。

亦请参阅

有关辅助窗口记录的说明请参阅 4-73 页中的“辅助窗口记录”一节。

低级例程

本节介绍低级例程，它们被高级窗口管理器例程调用。通常，您的应用程序不需要使用这些例程。

CheckUpdate

事件管理器使用 CheckUpdate 函数扫描窗口列表，搜索需要更新的窗口。

```
FUNCTION CheckUpdate (VAR theEvent: EventRecord): Boolean;
```

theEvent　窗口需要更新时应填写的事件记录。

说明

CheckUpdate函数从前向后扫描窗口列表，搜索需要更新的可见窗口(即，更新区域非空的可见窗口)。如果找到一个窗口，其窗口记录包含一个图片间接指针，该函数就将自己重新绘制窗口，然后它继续在列表中进行搜索。如果找到一个窗口，其更新区域非空，而且窗口记录不包含一个图片间接指针，该函数就在 theEvent 参数中储存一个更新事件，并返回 TRUE。如果它没有找到这种窗口，就返回 FALSE。

事件管理器是平常唯一调用 CheckUpdate 的软件。

ClipAbove

为了显示窗口，窗口管理器使用ClipAbove过程确定窗口管理器端口的剪辑区域。

```
PROCEDURE ClipAbove (window: WindowPeek);
```

window　　窗口的窗口记录的一个指针。

说明

ClipAbove过程把窗口管理器端口的剪辑区域设定为和当前剪辑区域相交的桌面面积，减去在指定窗口前面的所有窗口结构区域。

ClipAbove 过程从全局变量 GrayRgn 检索桌面区域。

SaveOld

窗口管理器使用 SaveOld 过程存储窗口的当前结构和内容区域，为更新窗口作准备。

```
PROCEDURE SaveOld (window: WindowPeek);
```

window　窗口的完全窗口记录的一个指针。

说明

SaveOld 过程为 DrawNew 过程存储指定窗口的当前结构区域和内容区域。每次调用 SaveOld 后，必须调用 DrawNew 作为平衡。

DrawNew

窗口管理器使用 DrawNew 过程抹除及更新更改的窗口区域。

```
PROCEDURE DrawNew (window: WindowPeek; update: Boolean);
```

window　窗口的窗口记录的一个指针。

update　确定区域是否被更新的一个布尔值。

说明

DrawNew过程抹去窗口的结构区域和内容区域的一部分，它们是窗口原来状态的一部分或窗口新状态的一部分，但不会同时有原来状态和新状态。这可以表示为：

```
(OldStructure XOR NewStructure) UNION (OldContent XOR NewContent)
```

如果更新参数被设定为 TRUE，DrawNew 还更新抹去的区域。

警告

SaveOld 和 DrawNew 在 Pascal 中不能嵌套。

汇编语言信息

在汇编语言中，如果您存储并还原全局变量 OldStructure 和 OldContent 的值，就可以把 SaveOld 和 DrawNew 嵌套。

PaintOne

窗口管理器使用PaintOne过程在桌面环境中重新绘制一个窗口的无效的、暴露的部分。

```
PROCEDURE PaintOne (window: WindowPeek; clobberedRgn: RgnHandle);
```

window　　　　窗口的窗口记录的一个指针。

clobberedRgn　变成无效的区域的一个间接指针。

说明

PaintOne过程对指定窗口的无效部分和它上面的所有窗口"涂色"。PaintOne绘制在clobberedRgn中的全部窗口框架，如果内容区域的某个部分暴露了，它还抹去暴露的面积（涂上背景图案），并把它添加至窗口的更新区域。如果窗口参数的值是NIL，窗口就是桌面环境，PaintOne用桌面图案进行涂色。

汇编语言信息

全局变量SaveUpdate和PaintWhite是PaintOne使用的标志。在正常情况下，两个标志都是设定的。清除SaveUpdate将防止clobberedRgn被添加至窗口的更新区域。清除PaintWhite将防止clobberedRgn在被添加至窗口的更新区域之前就抹除。（例如，若窗口的背景图案不是桌面环境的背景图案，这是有用的。）窗口管理器周期性地设定这两个标志，所以，您每次需要清除它们时，相应地做出必要的清除。

PaintBehind

窗口管理器使用PaintBehind过程重新绘制窗口列表中的一系列窗口。

```
PROCEDURE PaintBehind (startWindow: WindowPeek;
                       clobberedRgn: RgnHandle);
```

startWindow　窗口的窗口记录的一个指针。

clobberedRgn　变成无效的区域的一个间接指针。

说明

PaintBehind过程调用PaintOne，把startWindow和SurfWriter后面的所有窗口剪辑到clobberedRgn。

汇编语言信息

因为PaintBehind在调用PaintOne之前清除全局变量PaintWhite，clobberedRgn没有被抹去。调用PostingOptions后，全局变量PaintWhite被重新设定。

CalcVis

窗口管理器使用CalcVis过程计算一个窗口的可视区域。

```
PROCEDURE CalcVis (window: WindowPeek);
```

window　　该窗口的窗口记录的一个指针。

说明

CalcVis过程计算指定窗口的可视区域，它先计算窗口的内容区域，然后减去该窗口前面的每个窗口的结构区域。

CalcVisBehind

窗口管理器使用CalcVisBehind 过程计算一系列窗口的可视区域。

```
PROCEDURE CalcVisBehind (startWindow: WindowPeek;
                             clobberedRgn: RgnHandle);
```

startWindow　　窗口的窗口记录的一个指针。

clobberedRgn　　变成无效的桌面区域的一个间接指针。

说明

CalcVisBehind过程计算以下窗口的可视区域 startWindow参数指定的窗口和所有在startWindow后面并和clobberedRgn相交的窗口。该过程在PaintBehind之后调用。

应用程序定义的例程

本节介绍窗口定义函数。窗口管理器提供的窗口定义函数处理标准窗口类型，这些类型已经在4-8页的“窗口类型”中说明过。

窗口定义函数

如果您的应用程序定义它自己的窗口类型，您必须提供自己的窗口定义函数处理它们。应把您的定义函数作为‘WDEF’类型的资源储存，资源标识符从 128 到 4096。(窗口定义函数资源标识符为 0 和 1 的是系统设定窗口，从 2 到 127 的资源标识符为苹果公司保留。)

您的窗口定义函数可以支持多达 16 个变量代码，它们被标识为 0 到 15。若要请求使用您自己的窗口类型，您应指定窗口定义的 ID 值，它的高端 12 位是窗口定义函数的资源标识符，低端 4 位是变量代码。这样，对一个给定的资源标识符和变量代码，窗口定义标识符为（16* 资源标识符）+（变量代码）

当您创建一个窗口时，窗口管理器调用资源映象存取窗口定义函数。资源管理器把窗口定义函数读入存储器，并返回它的一个间接指针。窗口管理器在窗口记录的 windowDefProc 域中储存这个间接指针。(如果使用的是 24 位地址，窗口管理器就把变量代码储存在windowDefProc 域的低端 4 位中；如果使用的是 32 位地址，窗口管理器就把变量代码储存在其它地方。)以后，当需要执行类型有关的动作时，窗口管理器调用窗口定义函数，并把变量代码作为参数传送给它。

MyWindow

窗口定义函数负责执行下列任务：绘制窗口框架；报告鼠标按下事件发生的区域；计算窗口的结构区域和内容区域；绘制尺寸格；当用户拖拽尺寸格时调整窗口框架的大小；以及执行所有自定的初始化或处理任务。

您可以为您的窗口定义函数起任何名称。它有 4 个参数并返回一个结果代码：

```
FUNCTION MyWindow (varCode: Integer; theWindow: WindowPtr;
                   message: Integer; param: LongInt): LongInt;
```

varCode　　窗口的变量代码。

theWindow　　窗口的窗口记录的一个指针。

message　要执行的任务的代码。message 参数包含下列这些值之一:

```
CONST
    wDraw        = 0;    {draw window frame}
    wHit         = 1;    {report where mouse-down event }
                         { occurred}
    wCalcRgns    = 2;    {calculate strucRgn and contRgn}
    wNew         = 3;    {perform additional }
                         { initialization}
    wDispose     = 4;    {perform additional disposal }
                         { tasks}
    wGrow        = 5;    {draw grow image during resizing}
    wDrawGIcon   = 6;    {draw size box and scroll bar }
                         { outline}
```

下面各段详细解释上面的每个任务。

param 和 message 参数指定的任务有关的数据。如果该任务不要求数据，就忽略这个参数。

您的窗口定义函数执行由 message 参数指定的任何任务，如果需要，还返回一个函数结果。如果执行的任务不需要结果代码，就返回 0。函数的输入点必须在函数的开始。

您可以把各种任务作为子例程设置在窗口定义函数中，但是并不要求您必须这样做。

绘制窗口框架

当您收到 wDraw 消息时，就在当前图形端口中绘制窗口框架，当前图形端口是窗口管理器的端口。

您必须作一些核对，以便确定如何精确地绘制框架。如果窗口记录中 visible 字段的值是 FALSE，您不应该做任何事情；如果不是 FALSE，您应该检验 param 参数和窗口记录中的状态标志:

- 如果 param 的值是 0，就绘制整个窗口框架
- 如果 param 的值是 0，而且窗口记录中的 hilited 域是 TRUE，就醒目显示该框架，示窗口是激活的
- 如果窗口记录中的 goAwayFlag 域的值也是 TRUE，在窗口框架上绘制一个关闭格
- 如果窗口记录中的 spareFlag 域的值也是 TRUE，在窗口框架上绘制一个缩放格
- 如果 param 参数的值是 wInGoAway，就醒目显示窗口的关闭格或取消它的醒目示。4-46 页中的图 4-19 显示了醒目显示的关闭格和没有醒目显示的关闭格，它们是由窗口管理器的窗口定义函数绘制的

- 如果param参数的值是wInZoom，就醒目显示窗口的缩放格或取消它的醒目显示。47页中的图 4-20显示了醒目显示的缩放格和没有醒目显示的缩放格，它们是由窗口管理器的窗口定义函数绘制的

附注:

当窗口管理器调用一个message代码为 wDraw 的窗口定义函数时，它把Integer类型的值储存在param参数中，而且不清除高位字。当处理wDraw消息时，仅使用param参数的低位字。

窗口框架通常包括窗口的标题，但并不必须包括。窗口标题应该采用系统字体和系统字体大小显示。窗口管理器端口已经设定为使用系统字体和系统字体大小。

当设计一个包含窗口标题的标题栏时，应允许在垂直方向至少有 16 个象素高，这样才能支持在系统字体最小为 12 点的语系系统中的本地化。

附注:

在窗口结构区域以外所绘制的内容都是不可见的。

返回鼠标按下事件的区域

当您收到一个 wHit 消息时，您必须确定按下鼠标按钮时光标的位置。wHit 消息伴随着param 参数中的鼠标位置(以全局坐标表示)。垂直坐标在该参数的高位字，水平坐标在低位字。您返回下列常数之一:

```
CONST
    wNoHit        = 0;    {none of the following}
    wInContent    = 1;    {in content region (except grow, if
                          active)}
    wInDrag       = 2;    {in drag region}
    wInGrow       = 3;    {in grow region (active window only)}
    wInGoAway     = 4;    {in go-away region (active window
                          only)}
    wInZoomIn     = 5;    {in zoom box for zooming in (active
                          window }{ only)}
    wInZoomOut    = 6;    {in zoom box for zooming out (ac
                          tive window }{ only)}
```

返回值wNoHit可能（但不一定）意味着这个点不在窗口内。例如，如果这个点在窗口框架上但不在标题栏上，标准的窗口定义函数返回wNoHit。

只有当窗口是当前活动窗口时，才返回wInGrow、wInGoAway、wInZoomIn和wInZoomOut，因为按照惯例，非当前活动窗口的尺寸格、关闭格和缩放格是不绘制的。例如，在一个非现用的文稿窗口中，如果一个鼠标按下事件发生在标题栏上窗口激活时放置关闭格的地方，返回的常数仍然是wInDrag。

计算区域

当您收到wCalcRgn消息时，您就根据当前图形端口的端口矩形计算窗口的结构区域和内容区域。这些区域的间接指针在窗口记录中的strucRgn和contRgn域中，以全局坐标表示。只有当窗口是可见窗口时，窗口管理器才请求这个操作。

警告

当您为自己定义的类型的窗口计算区域时，不要更改窗口图形端口的剪辑区域和可视区域。窗口管理器和QuickDraw为您执行这些任务。更改剪辑区域或可视区域可能会破坏其它窗口。

初始化新的窗口

当您收到wNew消息时，可以执行任何可能需要的、取决于类型的窗口初始化。例如，如果内容区域的形状很特殊，您可能会为区域分配存储器，并把区域的间接指针储存在窗口记录的dataHandle域中。标准文稿窗口的初始化例程创建wStateData记录储存缩放数据。

处理窗口

当您收到wDispose消息时，可以执行处理窗口所必须的任何附加的任务。例如，您可能会释放初始化例程分配的存储器。标准文稿窗口的处理例程处理wStateData记录。

调整窗口的大小

当您收到wGrow消息时，就绘制窗口的一个增长图像。和wGrow消息一起收到的还有用全局坐标表示的矩形的一个指针，它的左上角和窗口图形端口的端口矩形对齐。您的增长图像应该处在矩形的内部。当用户拖拽鼠标时，窗口管理器重复地发送wGrow消息，这样您就可以更改增长图像，使它和更改的鼠标位置匹配。

您应该在当前图形端口中以当前画笔图案和画笔模式绘制增长图像，这个端口是窗口管理器端口。这些是和Macintosh用户界面准则一致的设置（和grey及notPatXor一样）。

标准文稿窗口的增长例程绘制窗口的一个虚线（灰）轮廓线，以及标题栏、尺寸格和滚动杆的定界线。

绘制尺寸格

当您收到wDrawGIcon消息时，如果窗口是当前活动窗口，您就在内容区域绘制尺寸格；如果窗口是非当前活动窗口，就绘制能够表示不能更改该窗口大小的任何适当的文字。

附注:

如果尺寸格不在内容区域中，而是在窗口框架上，您不需要做任何事情响应 wDrawGIcon 消息，只需响应 wDraw 消息绘制尺寸格。

为现用文稿窗口绘制尺寸格的例程在窗口图形端口的端口矩形的右下角绘制大小框。它还绘制尺寸格和滚动杆面积的定界线。对一个非当前活动窗口，它抹除大小框，并绘制定界线。

资源

本节介绍窗口管理器使用的资源。

- 描述窗口特征的 'WIND' 资源
- 保持窗口定义函数的 'WDEF' 资源
- 定义窗口框架和醒目显示所使用的彩色的 'wctb' 资源

窗口资源

通常您需要对应用程序创建的每一种类型的窗口定义一个窗口资源。图4-24显示了一个编译的窗口（ 'WIND' ）资源。

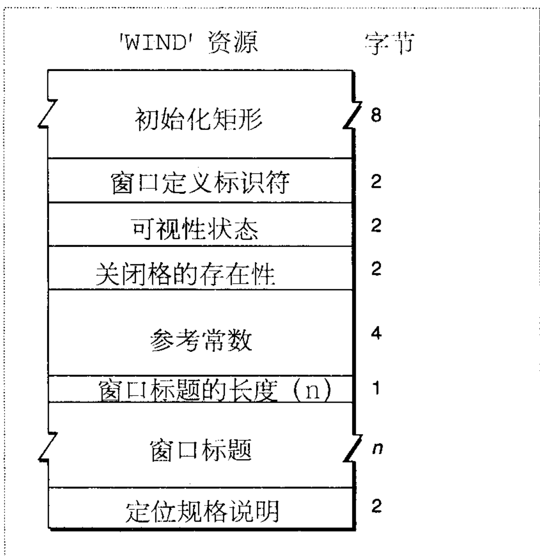


图 4-24 编译的窗口（ 'WIND' ）资源的结构

一个编译的窗口资源包含下列元素:

- 以全局坐标表示的一个矩形的左上角和右下角,该矩形定义窗口内容区域的初始大小和位置。您的应用程序可以在显示窗口前，通过编程或本节后面说明的可选定位代码更改这个矩形
- 窗口的定义标识符，它结合了处理窗口的窗口定义函数的资源标识符和一个可选的变量代码。窗口定义函数的资源标识符和变量代码在一起定义了一种窗口类型。窗口定义函数的资源标识符放置在定义标识符的高端 12 位上。资源标识符从 0 到 127 的窗口定义函数是为苹果公司保留的。可选的变量代码放置在定义标识符的低端 4 位上

如果您使用的是一个标准窗口类型（在4-8 页的“窗口类型”中作了说明），定义标识符就是下列窗口类型常数之一:

```
CONST
     documentProc  = 0;      {movable, sizable window, }
                             { no zoom box}
     dBoxProc      = 1;      {alert box or modal dialog box}
     plainDBox     = 2;      {plain box}
     altDBoxProc   = 3;      {plain box with shadow}
     noGrowDocProc = 4;      {movable window, no size box or }
                             { zoom box}
     movableDBoxProc= 5;     {movable modal dialog box}
     zoomDocProc   = 8;      {standard document window}
     zoomNoGrow    = 12;     {zoomable, nonresizable window}
     rDocProc      = 16;     {rounded-corner window}
```

您也可以通过指定两个常数的和 movableDBoxProc + zoomDocProc，把缩放格添加至可移动定模式对话框，但是最好在任何对话框中都不要添加缩放格。

您可以通过添加下列整数之一，控制圆角窗口的圆角曲速率:

窗口定义标识符	曲速率直径
rDocProc	16, 16
rDocProc + 2	4, 4
rDocProc + 4	6, 6
rDocProc + 6	10, 10

- 确定窗口是否可见的一个样板说明。这个特征只控制窗口管理器是否显示窗口，而不管窗口是否在幕上可以看到。(例如，如果一个可见窗口完全被其它窗口覆盖，它也是“可见”的，即使用户实际上看不到它。）通常您以不可见状态创建一个新窗口，建立它的内容区域，然后才显示完整的窗口。

■ 确定窗口是否有一个关闭格的样板说明。窗口管理器在绘制窗口框架的同时，绘制关闭格。在第二个域指定的窗口类型确定了一个窗口是否支持关闭格；该字段决定了关闭格是否存在

■ 一个参考常数，您的应用程序可以使用它储存需要的任何数据。当它建立一个新的窗口记录时，窗口管理器在refCon域储存您在窗口资源的第五个元素指定的任何值。您也可以在这里放置一个位置标志符，然后使用SetWRefCon过程设定refCon域

■ 指定窗口标题的一个字符串。字符串的第一个字节指定了字符串的长度（即，标题中的字符数加上 1 个记录长度的字节），该长度以字节数为单位

■ 一个可选的定位规格说明，它优先于第一域的矩形建立的窗口位置。定位值可以是这里列出的一个整数。在这些常数的名称中，术语的意义如下：

center

在垂直和水平方向，相对于整个屏幕或另一个窗口都居中（如果一个较宽的窗口要相对于它前面的一个较窄的窗口居中，较宽的窗口在左边固定，较窄的窗口在宽窗口中居中）

stagger

位于上一个窗口 10 个象素以下，10 个像素以右（在相对屏幕固定时，第一个窗口在屏幕的左边，恰好在菜单栏下的位置，后继的窗口相对于第一的窗口放置）

alert position

水平居中，垂直方向放置在“警告位置”，大约窗口或屏幕的 1/5 在新窗口的上面，其它在下面。

parent window

用户上一次工作的窗口

资源第七个元素可以包含下列常数指定的一个值：

```
CONST    noAutoCenter        = 0x0000;      {use initial }
                                            { location}
         centerMainScreen    = 0x280A;      {center on main }
                                            { screen}
         alertPositionMainScreen= 0x300A;{place in alert }
                                            { position on main }
                                            { screen}
         staggerMainScreen = 0x380A;        {stagger on main }
                                            { screen}
         centerParentWindow= 0xA80A;        {center on parent }
                                            { window}
```

```
alertPositionParentWindow       = 0xB00A;{place in alert }
                                         { position on }
                                         { parent window}
staggerParentWindow             = 0xB80A;{stagger relative }
                                         { to parent window}
centerParentWindowScreen        = 0x680A;{center on parent }
                                         { window screen}
alertPositionParentWindowScreen = 0x700A;{place in alert }
                                         { position on }
                                         { parent window }
                                         { screen}
staggerParentWindowScreen       = 0x780A;{stagger on parent}
                                         { window screen}
```

当用户在创建新文稿或当您在处理您自己的对话框和警告栏时，定位常数是很方便的。但是，当您创建一个新窗口显示以前存储的文稿时，您应该在与以前的窗口（即文稿最后存储时所占据的窗口）相同的矩形中显示新窗口。若想了解更多的信息，请参阅 4 - 30 页的“在桌面上定位文稿窗口”。

使用 GetNewCWindow 或 GetNewWindow 函数来读取 'WIND' 资源。这两个函数均能创建一个新的窗口记录并根据在 'WIND' 资源中指定的值填写记录。

窗口定义函数资源

窗口定义函数可以被当作 'WDEF' 类型的资源存储起来。'WDEF' 资源其实就是窗口定义函数的可执行代码。

窗口管理器提供的两个标准的窗口定义函数使用的是资源标识符 0 和 1。

窗口彩色表资源

您可以把窗口色彩表指定为 'wctb' 类型的资源。

通常您不应该定义您自己的窗口彩色表，除非您需要对窗口框架的色彩或文字醒目显示进行一些特别的控制。若要在创建窗口时给窗口指派一个色彩表，需要提供一个与 'WIND' 资源具有相同的资源标识符的窗口色彩表（'wctb'）资源（您根据 'WIND ' 资源创建此窗口）。

窗口色彩表资源是窗口色彩表数据结构的准确图像。图 4 - 25 给出了一个经过编译的 'wctb' 资源的内容。

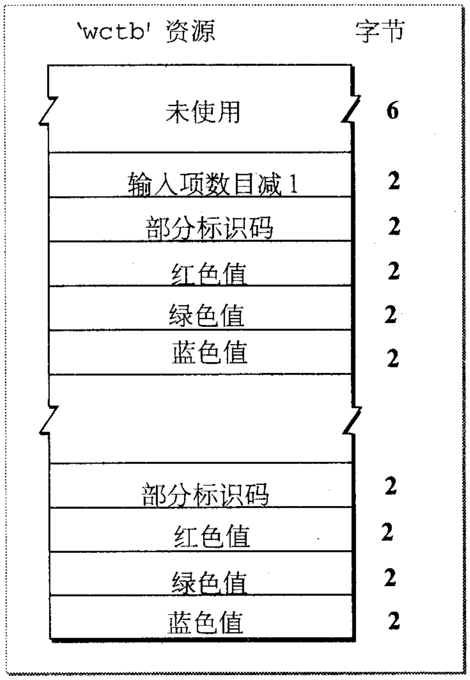


图 4 - 25　经过编译的窗口色彩表（'wctb'）资源的结构

一个编译后的窗口资源包含下列单元:

- 一个 6 字节长的未使用的域
- 一个整数，其值为资源中的输入项数目（即色彩说明记录的数目）减 1
- 一系列色彩规格记录，其中的每一个记录均包含一个 2 字节的部分标识码和 3 个 2 字节的色彩值。部分标识码是由下面这些常数所指定的一个整数

色彩值是指在每个窗口部分中的红色、绿色和蓝色的密度。（请参阅《Inside Macintosh：Imaging for a de语系ion of RGB color》）。

窗口管理器总结

Pascal 概述

常数

```
CONST
    {window types}
    documentProc  = 0;      {movable, sizable window, no zoom box}
    dBoxProc      = 1;      {alert box or modal dialog box}
    plainDBox     = 2;      {plain box}
    altDBoxProc   = 3;      {plain box with shadow}
    noGrowDocProc = 4;      {movable window, no size box or }
                            { zoom box}
    movableDBoxProc= 5;     {movable modal dialog box}
    zoomDocProc   = 8;      {standard document window}
    zoomNoGrow    = 12;     {zoomable, nonresizable window}
    rDocProc      = 16;     {rounded-corner window}
    {window kinds}
    dialogKind    = 2;      {dialog or alert box window}
    userKind      = 8;      {window created by the application}
    {part codes returned by FindWindow}
    inDesk        = 0;      {none of the following}
    inMenuBar     = 1;      {in menu bar}
    inSysWindow   = 2;      {in desk accessory window}
    inContent     = 3;      {anywhere in content region except size }
                            { box if window is active, }
                            { anywhere including size box if window }
                            { is inactive}
    inDrag        = 4;      {in drag (title bar) region}
    inGrow        = 5;      {in size box (active window only)}
    inGoAway      = 6;      {in close box}
    inZoomIn      = 7;      {in zoom box (window in standard state)}
    inZoomOut     = 8;      {in zoom box (window in user state)}
    {axis constraints on DragGrayRgn}
    noConstraint  = 0;      {no constraints}
    hAxisOnly     = 1;      {move on horizontal axis only}
    vAxisOnly     = 2;      {move on vertical axis only}
```

```
{window definition function wHit return codes}
wNoHit            = 0; {none of the following}
wInContent        = 1; {anywhere in content region exceptsize }
                       { box if window is active, }
                       { anywhere including size box ifwindow }
                       { is inactive}
wInDrag           = 2; {in drag (title bar) region}
wInGrow           = 3; {in size box (active window only)}
wInGoAway         = 4; {in close box}
wInZoomIn         = 5; {in zoom box (window in standardstate)}
wInZoomOut        = 6; {in zoom box (window in user state)}
{window color information table part codes}
wContentColor     = 0; {content region background}
wFrameColor       = 1; {window outline}
wTextColor        = 2; {window title and button text}
wHiliteColor      = 3; {reserved}
wTitleBarColor    = 4; {reserved}
wHiliteColorLight = 5; {lightest stripes in title bar }
                       { and lightest dimmed text}
wHiliteColorDark  = 6; {darkest stripes in title bar }
                       { and darkest dimmed text}
wTitleBarLight    = 7; {lightest parts of title bar back ground}
wTitleBarDark     = 8; {darkest parts of title bar background}
wDialogLight      = 9; {lightest element of dialog boxframe}
wDialogDark       = 10;{darkest element of dialog box frame}
wTingeLight       = 11;{lightest window tinging}
wTingeDark        = 12;{darkest window tinging}
                        {resource ID of desktop pattern}
deskPatID         = 16;
```

数据类型

```
TYPE                CWindowPtr    = CGrafPtr;
                    CWindowPeek   = ^CWindowRecord;
CWindowRecord =
RECORD
    port:           CGrafPort;    {window's graphics port}
    windowKind:     Integer;      {class of window}
    visible:        Boolean;      {visibility}
    hilited:        Boolean;      {highlighting}
    goAwayFlag:     Boolean;      {presence of close box}
    spareFlag:      Boolean;      {presence of zoom box}
    strucRgn:       RgnHandle;    {handle to structure region}
    contRgn:        RgnHandle;    {handle to content region}
    updateRgn:      RgnHandle;    {handle to update region}
    windowDefProc:Handle;         {handle to window definition function}
    dataHandle:     Handle;       {handle to window state data record}
    titleHandle:    StringHandle; {handle to window title}
    titleWidth:     Integer;      {title width in pixels}
    controlList:    ControlHandle;{handle to control list}
    nextWindow:     CWindowPeek;  {pointer to next window record in }
                                  { window list}
    windowPic:      PicHandle;    {handle to optional picture}
    refCon:         LongInt;      {storage available to your application}
END;
WindowPtr           = GrafPtr;
WindowPeek          = ^WindowRecord;
WindowRecord  =
RECORD                            {all fields have same use as }
                                  { in color window record}
    port:           GrafPort;     {window's graphics port}
    windowKind:     Integer;      {class of window}
    visible:        Boolean;      {visibility}
    hilited:        Boolean;      {highlighting}
    goAwayFlag:     Boolean;      {presence of close box}
    spareFlag:      Boolean;      {presence of zoom box}
    strucRgn:       RgnHandle;    {handle to structure region}
    contRgn:        RgnHandle;    {handle to content region}
    updateRgn:      RgnHandle;    {handle to update region}
    windowDefProc:Handle;         {handle to window definition function}
    dataHandle:     Handle;       {handle to window state data record}
```

```
    titleHandle:        StringHandle;       {handle to window title}
    titleWidth:         Integer;            {title width in pixels}
    controlList:        ControlHandle;      {handle to control list}
    nextWindow:         WindowPeek;         {pointer to next window record in}
                                            { window list}
    windowPic:          PicHandle;          {handle to optional picture}
    refCon:             LongInt;            {storage available to your appli
                                            cation}
END;
WStateDataPtr = ^WStateData;
WStateDataHandle = ^WStateDataPtr;
WStateData =                                {zoom state data record}
RECORD
    userState:                   Rect;      {size and location established
                                            by user}
    stdState:                    Rect;      {size and location established
                                            by application}
END;
WCTabPtr = ^WinCTab;
WCTabHandle = ^WCTabPtr;
WinCTab =                                   {window color information table}
RECORD
    wCSeed:                 LongInt;        {reserved}
    wCReserved:             Integer;        {reserved}
    ctSize:                 Integer;        {number of entries in table -1}
    ctTable:                ARRAY [0..4] OF ColorSpec;
                            {array of color specification records}
END;
ColorSpec      =
RECORD
    value:              Integer;            {part identifier}
    rgb:                RGBColor;           {RGB value}
END;
AuxWinHandle            = ^AuxWinPtr;
AuxWinPtr               = ^AuxWinRec;
AuxWinRec               =                   {auxiliary window record}
RECORD
    awNext:             AuxWinHandle;       {handle to next record}
    awOwner:            WindowPtr;          {pointer to window}
    awCTable:           CTabHandle;         {handle to color table}
```

```
    dialogCItem:          Handle;       {storage used by Dialog Manager}
    awFlags:              LongInt;      {reserved}
    awReserved:           CTabHandle;   {reserved}
    awRefCon:             LongInt;      {reference constant, for }
                                        { use by application}
END;
```

窗口管理器例程

初始化窗口管理器

```
PROCEDURE InitWindows;
```

创建窗口

```
FUNCTION GetNewCWindow (windowID: Integer; wStorage: Ptr;
behind: WindowPtr): WindowPtr;
FUNCTION GetNewWindow  (windowID: Integer; wStorage: Ptr;
behind: WindowPtr): WindowPtr;
FUNCTION NewCWindow    (wStorage: Ptr; boundsRect: Rect;
title: Str255; visible: Boolean;
procID: Integer; behind: WindowPtr;
goAwayFlag: Boolean;
refCon: LongInt): WindowPtr;
FUNCTION NewWindow     (wStorage: Ptr; boundsRect: Rect;
title: Str255; visible: Boolean;
theProc: Integer; behind: WindowPtr;
goAwayFlag: Boolean;
refCon: LongInt): WindowPtr;
```

命名窗口

```
PROCEDURE SetWTitle    (theWindow: WindowPtr; title: Str255);
PROCEDURE GetWTitle    (theWindow: WindowPtr; VAR title: Str255);
```

显示窗口

```
PROCEDURE DrawGrowIcon (theWindow: WindowPtr);
PROCEDURE SelectWindow (theWindow: WindowPtr);
PROCEDURE ShowWindow   (theWindow: WindowPtr);
PROCEDURE HideWindow   (theWindow: WindowPtr);
PROCEDURE ShowHide     (theWindow: WindowPtr; showFlag: Boolean);
PROCEDURE HiliteWindow (theWindow: WindowPtr; fHilite: Boolean);
PROCEDURE BringToFront (theWindow: WindowPtr);
PROCEDURE SendBehind   (theWindow, behindWindow: WindowPtr);
```

检索窗口信息

```
FUNCTION FindWindow    (thePoint: Point;VAR theWindow: WindowPtr): Integer;
FUNCTION FrontWindow   : WindowPtr;
```

移动窗口

```
PROCEDURE DragWindow (theWindow: WindowPtr; startPt: Point; boundsRect: Rect);
PROCEDURE MoveWindow (theWindow: WindowPtr; hGlobal, vGlobal: Integer; front:
                      Boolean);
FUNCTION DragGrayRgn (theRgn: RgnHandle; startPt: Point;
limitRect, slopRect: Rect; axis: Integer;    actionProc: ProcPtr): LongInt;
FUNCTION PinRect     (theRect: Rect; thePt: Point): LongInt;
```

更改窗口尺寸

```
FUNCTION GrowWindow (theWindow: WindowPtr; startPt: Point; sizeRect: Rect):
LongInt;
PROCEDURE SizeWindow (theWindow: WindowPtr; w, h: Integer;fUpdate: Boolean);
```

缩放窗口

```
FUNCTION TrackBox    (theWindow: WindowPtr; thePt: Point;partCode: Integer):
                      Boolean;
PROCEDURE ZoomWindow (theWindow: WindowPtr; partCode: Integer; front:
                      Boolean);
```

关闭和解除窗口内存空间

```
FUNCTION TrackGoAway     (theWindow: WindowPtr; thePt: Point): Boolean;
PROCEDURE CloseWindow    (theWindow: WindowPtr);
PROCEDURE DisposeWindow  (theWindow: WindowPtr);
```

保留更新区域

```
PROCEDURE BeginUpdate  (theWindow: WindowPtr);
PROCEDURE EndUpdate    (theWindow: WindowPtr);
PROCEDURE InvalRect    (badRect: Rect);
PROCEDURE InvalRgn     (badRgn: RgnHandle);
PROCEDURE ValidRect    (goodRect: Rect);
PROCEDURE ValidRgn     (goodRgn: RgnHandle);
```

设定和检索其它窗口特性

```
PROCEDURE SetWindowPic(theWindow: WindowPtr; Pic: PicHandle);
FUNCTION GetWindowPic  (theWindow: WindowPtr): PicHandle;
PROCEDURE SetWRefCon   (theWindow: WindowPtr; data: LongInt);
FUNCTION GetWRefCon    (theWindow: WindowPtr): LongInt;
FUNCTION GetWVariant   (theWindow: WindowPtr): Integer;
```

操纵桌面环境

```
PROCEDURE SetDeskCPat  (deskPixPat: PixPatHandle);
FUNCTION GetGrayRgn    : RgnHandle;
PROCEDURE GetCWMgrPort     (VAR wMgrCPort: CGrafPtr);
PROCEDURE GetWMgrPort (VAR wPort: GrafPtr);
```

操纵窗口彩色信息

```
PROCEDURE SetWinColor  (theWindow: WindowPtr; newColorTable: WCTabHandle);
FUNCTION GetAuxWin         (theWindow: WindowPtr;
VAR awHndl: AuxWinHandle): Boolean;
```

低级例程

```
FUNCTION CheckUpdate   (VAR theEvent: EventRecord): Boolean;
PROCEDURE ClipAbove    (window: WindowPeek);
PROCEDURE SaveOld      (window: WindowPeek);
PROCEDURE DrawNew          (window: WindowPeek; update: Boolean);
PROCEDURE PaintOne         (window: WindowPeek; clobberedRgn: RgnHandle);
PROCEDURE PaintBehind  (startWindow: WindowPeek;clobberedRgn: RgnHandle);
PROCEDURE CalcVis      (window: WindowPeek);
PROCEDURE CalcVisBehind(startWindow: WindowPeek;
clobberedRgn: RgnHandle);
```

应用程序定义例程

窗口定义函数

```
FUNCTION MyWindow          (varCode: Integer; theWindow: WindowPtr; message:
Integer;                           param:LongInt): LongInt;
```

C 概述

常数

```
enum {
    /*window types*/
    documentProc          = 0, /*movable, sizable window, no zoom box*/
    dBoxProc              = 1, /*alert box or modal dialog box*/
    plainDBox             = 2, /*plain box*/
    altDBoxProc           = 3, /*plain box with shadow*/
    noGrowDocProc         = 4, /*movable window, no size box or zoom box*/
    movableDBoxProc       = 5, /*movable modal dialog box*/
    zoomDocProc           = 8, /*standard document window*/
    zoomNoGrow            = 9, /*zoomable, nonresizable window*/
    rDocProc              = 16,/*rounded-corner window*/

    /*window kinds*/
    dialogKind            = 2, /*dialog or alert box window*/
    userKind              = 8, /*window created by the application*/

    /*part codes returned by FindWindow*/
    inDesk                = 0, /*none of the following*/
    inMenuBar             = 1, /*in menu bar*/
    inSysWindow           = 2, /*in desk accessory window*/
    inContent             = 3, /*anywhere in content region except size
                                 box if*/
                                 /* window is active, anywhere including */
                                 /* size box if window is inactive*/
    inDrag                = 4, /*in drag (title bar) region*/
    inGrow                = 5, /*in size box (active window only)*/
    inGoAway              = 6, /*in close box*/
    inZoomIn              = 7, /*in zoom box (window in standard state)*/
    inZoomOut             = 8  /*in zoom box (window in user state)*/
};

enum {
    /*axis constraints on DragGrayRgn*/
    noConstraint          = 0,     /*no constraints*/
    hAxisOnly             = 1,     /*move on horizontal axis only*/
    vAxisOnly             = 2      /*move on vertical axis only*/
};
```

```
enum {
    /*window definition function task codes*/
    wDraw               = 0,     /*draw window frame*/
    wHit                = 1,     /*report where mouse-down occurred*/
    wCalcRgns           = 2,     /*calculate strucRgn and contRgn*/
    wNew                = 3,     /*perform additional initialization*/
    wDispose            = 4,     /*perform additional disposal tasks*/
    wGrow               = 5,     /*draw grow image during resizing*/
    wDrawGIcon          = 6,     /*draw size box and scroll bar outline*/

    /*window definition function wHit return codes*/
    wNoHit              = 0,     /*none of the following*/
    wInContent          = 1,     /*in content region (except grow, if active)*/
    wInDrag             = 2,     /*in drag region*/
    wInGrow             = 3,     /*in grow region (active window only)*/
    wInGoAway           = 4,     /*in go-away region (active window only)*/
    wInZoomIn           = 5,     /*in zoom box for zooming in (active window */
                                 /* only)*/
    wInZoomOut          = 6,     /*in zoom box for zooming out (active window*/
                                 /* only)*/
    deskPatID           = 16,    /*resource ID of desktop pattern*/

    /*window color information table part codes*/
    wContentColor       = 0,     /*the background of the window's */
                                 /* content region*/
    wFrameColor         = 1,     /*the window outline*/
    wTextColor          = 2,     /*window title and text in buttons*/
    wHiliteColor        = 3,     /*reserved*/
    wTitleBarColor      = 4,     /*reserved*/
    wHiliteColorLight   = 5,     /*lightest stripes in title bar */
                                 /* and lightest dimmed text*/
    wHiliteColorDark    = 6,     /*darkest stripes in title bar */
                                 /* and darkest dimmed text*/
    wTitleBarLight      = 7,     /*lightest parts of title bar background*/
    wTitleBarDark       = 8,     /*darkest parts of title bar background*/
    wDialogLight        = 9,     /*lightest element of dialog box frame*/
    wDialogDark         = 10,    /*darkest element of dialog box frame*/
    wTingeLight         = 11,    /*lightest window tinging*/
    wTingeDark          = 12     /*darkest window tinging*/
};
```

数据类型

```
struct CWindowRecord {
  CGrafPort          port;              /*window's graphics port*/
    short            windowKind;        /*class of the window*/
    Boolean          visible;           /*visibility*/
    Boolean          hilited;           /*highlighting*/
    Boolean          goAwayFlag;        /*presence of close box*/
    Boolean          spareFlag;         /*presence of zoom box*/
    RgnHandle        strucRgn;          /*handle to structure region*/
    RgnHandle        contRgn;           /*handle to content region*/
    RgnHandle        updateRgn;         /*handle to update region*/
    Handle           windowDefProc;     /*handle to window definition */
                                        /* function*/
    Handle           dataHandle;        /*handle to window state data
                                         record*/
    StringHandle     titleHandle;       /*handle to window title*/
    short            titleWidth;        /*title width in pixels*/
    ControlHandle controlList;          /*handle to control list*/
    struct CWindowRecord *nextWindow;   /*next window in window list*/
    PicHandle        windowPic;         /*handle to optional picture*/
    long             refCon;            /*storage available to your */
                                        /* application*/
};
typedef struct CWindowRecord CWindowRecord;
typedef CWindowRecord *CWindowPeek;
struct WindowRecord {
    GrafPort         port;              /*window's graphics port*/
    short            windowKind;        /*class of the window*/
    Boolean          visible;           /*visibility*/
    Boolean          hilited;           /*highlighting*/
    Boolean          goAwayFlag;        /*presence of close box*/
    Boolean          spareFlag;         /*presence of zoom box*/
    RgnHandle        strucRgn;          /*handle to structure region*/
    RgnHandle        contRgn;           /*handle to content region*/
    RgnHandle        updateRgn;         /*handle to update region*/
    Handle           windowDefProc;     /*handle to window definition */
                                        /* function*/
    Handle           dataHandle;        /*handle to window state data
                                        record*/
    StringHandle     titleHandle;       /*handle to window title*/
    short            titleWidth;        /*title width in pixels*/
    ControlHandle controlList;          /*handle to window's control list*/
    struct WindowRecord*nextWindow;     /*next window in window list*/
```

```
    PicHandle              windowPic;          /*handle to optional picture*/
    long                   refCon;             /*reference constant*/
};
typedef struct WindowRecord WindowRecord;
typedef WindowRecord *WindowPeek;
struct WStateData {
    Rect      userState;                       /*user state*/
    Rect      stdState;                        /*standard state*/
};
typedef struct WStateData WStateData;
typedef WStateData *WStateDataPtr, **WStateDataHandle;
struct AuxWinRec {
    struct AuxWinRec       **awNext;           /*handle to next record*/
    WindowPtr              awOwner;            /*pointer to window */
    CTabHandle             awCTable;           /*handle to color table*/
    Handle                 dialogCItem;        /*storage used by Dialog Manager*/
    long                   awFlags;            /*reserved*/
    CTabHandle             awReserved;         /*reserved*/
    long                   awRefCon;           /*reference constant, for use by*/
                                               /* application*/
};
typedef struct AuxWinRec AuxWinRec;
typedef AuxWinRec *AuxWinPtr, **AuxWinHandle;
struct WinCTab {
    long                   wCSeed;             /*reserved*/
    short                  wCReserved;         /*reserved*/
    short                  ctSize;             /*number of entries in table -1*/
    ColorSpec              ctTable[5];         /*array of color specification
                                               records*/
};
typedef struct WinCTab WinCTab;
typedef WinCTab *WCTabPtr, **WCTabHandle;
```

窗口管理器例程

初始化窗口管理器

```
pascal void InitWindows(void);
```

创建窗口

```
pascal WindowPtr GetNewCWindow
(short windowID, void *wStorage,    WindowPtr behind);
pascal WindowPtr GetNewWindow
(short windowID, void *wStorage,WindowPtr behind);
pascal WindowPtr NewCWindow
(void *wStorage, const Rect *boundsRect,ConstStr255Param title,
Boolean visible,short procID, WindowPtr behind,Boolean goAwayFlag, long refCon);
pascal WindowPtr NewWindow
(void *wStorage, const Rect *boundsRect,ConstStr255Param title,
Boolean visible,short theProc, WindowPtr behind,Boolean
                goAwayFlag, long refCon);
```

命名窗口

```
pascal void SetWTitle  (WindowPtr theWindow, ConstStr255Param title);
pascal void GetWTitle  (WindowPtr theWindow, Str255 title);
```

Displaying Windows

```
pascal void DrawGrowIcon    (WindowPtr theWindow);
pascal void SelectWindow    (WindowPtr theWindow);
pascal void ShowWindow      (WindowPtr theWindow);
pascal void HideWindow      (WindowPtr theWindow);
pascal void ShowHide        (WindowPtr theWindow, Boolean showFlag);
pascal void HiliteWindow    (WindowPtr theWindow, Boolean fHilite);
pascal void BringToFront    (WindowPtr theWindow);
pascal void SendBehind      (WindowPtr theWindow, WindowPtr behindWindow);
Retrieving Mouse Information
pascal short FindWindow    (Point thePoint, WindowPtr *theWindow);
pascal WindowPtr FrontWindow(void);
```

Moving Windows

```
pascal void DragWindow (WindowPtr theWindow, Point startPt,const Rect
                       *boundsRect);
pascal void MoveWindow (WindowPtr theWindow, short hGlobal,short vGlobal,
                       Boolean front);
```

```
pascal long DragGrayRgn(RgnHandle theRgn, Point startPt,const Rect
*boundsRect,const Rect *slopRect,short axis, DragGrayRgnProcPtr actionProc);
pascal long PinRect       (const Rect *theRect, Point *thePt);
```

更改窗口尺寸

```
pascal long GrowWindow (WindowPtr theWindow, Point startPt,const Rect *bBox);
pascal void SizeWindow (WindowPtr theWindow, short w, short h,Boolean fUpdate);
```

缩放窗口

```
pascal Boolean TrackBox(WindowPtr theWindow, Point thePt,short partCode);
pascal void ZoomWindow (WindowPtr theWindow, short partCode,Boolean front);
```

关闭并解除窗口

```
pascal Boolean TrackGoAway (WindowPtr theWindow, Point thePt);
pascal void CloseWindow    (WindowPtr theWindow);
pascal void DisposeWindow  (WindowPtr theWindow);
```

保留更新区域

```
pascal void BeginUpdate(WindowPtr theWindow);
pascal void EndUpdate  (WindowPtr theWindow);
pascal void InvalRect  (const Rect *badRect);
pascal void InvalRgn   (RgnHandle badRgn);
pascal void ValidRect  (const Rect *goodRect);
pascal void ValidRgn   (RgnHandle goodRgn);
```

设定和检索其它窗口特性

```
pascal void SetWindowPic   (WindowPtr theWindow, PicHandle pic);
pascal PicHandle           GetWindowPic(WindowPtr theWindow);
pascal void  SetWRefCon    (WindowPtr theWindow, long data);
pascal long GetWRefCon     (WindowPtr theWindow);
pascal short GetWVariant   (WindowPtr theWindow);
```

操纵桌面环境

```
pascal void SetDeskCPat     (PixPatHandle deskPixPat);
#define GetGrayRgn()        (* (RgnHandle* 0X09EE))
pascal void GetCWMgrPort    (CGrafPtr *wMgrCPort);
pascal void GetWMgrPort     (GrafPtr *wPort);
```

操纵窗口彩色信息

```
pascal void SetWinColor     (WindowPtr theWindow,WCTabHandle newColorTable);
pascal Boolean GetAuxWin    (WindowPtr theWindow, AuxWinHandle *awHndl);
```

低级例程

```
pascal Boolean CheckUpdate (EventRecord *theEvent);
pascal void ClipAbove       (WindowPeek window;)
pascal void SaveOld         (WindowPeek window);
pascal void DrawNew         (WindowPeek window, Boolean update);
pascal void PaintOne        (WindowPeek window, RgnHandle clobberedRgn);
pascal void PaintBehind     (WindowPeek startWindow,RgnHandle clobberedRgn);
pascal void CalcVis         (WindowPeek window);
pascal void CalcVisBehind   (WindowPeek startWindow,RgnHandle clobberedRgn);
```

应用程序定义的例程

窗口定义函数

```
pascal long MyWindow   (short varCode, WindowPtr theWindow, short message,
                        long param);
```

汇编语言概述

数据类型

窗口记录和彩色窗口记录数据结构

0	windowPort	108 字节	窗口的图形端口
108	windowKind	字	窗口是怎样创建的
110	wVisible	字节	可视性状态
111	wHilited	字节	醒目显示状态
112	wGoAway	字节	有无关闭格
113	wZoom	字节	有无缩放格
114	structRgn	长型	结构区域间接指针
118	contRgn	长型	内容区域间接指针
122	updateRgn	长型	更新区域间接指针
126	windowDef	长型	窗口定义函数间接指针
130	wDataHandle	长型	窗口状态数据记录间接指针
134	wTitleHandle	长型	窗口标题间接指针
138	wTitleWidth	字	以象素为单位的标题宽度
140	wControlList	长型	窗口的控制列表间接指针
144	nextWindow	长型	窗口列表中下一个窗口的指针
148	windowPic	长型	更新图片的间接指针
152	wRefCon	长型	参考常数域

窗口状态数据结构

0	userState	8 字节	用户状态矩形
8	stdState	8 字节	标准状态矩形

窗口彩色信息表数据结构

0	ctSeed	长型	彩色信息表的 ID 号码
4	ctFlags	字	标志字
6	ctSize	字	输入项数目减 1
8	ctTable	变长	一系列色彩说明记录（每个 8 字节）

辅助窗口记录数据结构

0	awNext	长型	链中下一个窗口的间接指针
4	awOwner	长型	相关的窗口记录的指针
8	awCTable	长型	窗口彩色信息表间接指针
12	dialogCItem	长型	对话项目色彩结构的间接指针
16	awFlags	长型	QuickDraw 间接指针
20	awResrv	长型	保留
24	awRefCon	长型	用户常数

全局变量

AuxWinHead	辅助窗口列表起始端间接指针，
CurActivate	接收激活事件的窗口的指针。
CurDeactive	接收停用事件的窗口的指针。
DeskHook	给桌面涂色的过程的地址。
DeskPattern	桌面填充色（8字节）。
DragHook	在 TrackGoAway,TrackBox,DragWindow,GrowWindow,和 DragGrayRgn 运行期间执行的可选过程的地址。
DragPattern	被拖拽区域的轮廓线填充色。
GrayRgn	桌面区域间接指针。
OldContent	存储的内容区域间接指针。
OldStructure	存储的结构区域间接指针。
PaintWhite	表明在更新事件前是否把窗口涂成白色的标志（2字节）。
SaveUpdate	表明是否生成更新事件的标志（2字节）。
SaveVisRgn	存储的可视区域间接指针。
WindowList	窗口列表中第一个窗口的指针。
WMgrPort	窗口管理器端口指针。

第五章 控制管理器

控制部件介绍 5 - 4
　　按钮 5 - 5
　　复选框 5 - 5
　　选项按钮 5 - 6
　　弹出式菜单 5 - 6
　　滚动杆 5 - 7
　　其它控制部件 5 - 11
　　有效和无效控制部件 5 - 11
　　控制定义函数 5 - 14
关于控制管理器 5 - 14
使用控制管理器 5 - 15
　　创建和显示控制部件 5 - 15
　　　　创建按钮、复选框或选项按钮 5 - 17
　　　　创建滚动杆 5 - 21
　　　　创建弹出式菜单 5 - 25
　　　　更新控制部件 5 - 29
　　响应控制部件中的鼠标事件 5 - 30
　　　　确定控制部件中的鼠标按下事件 5 - 31
　　　　跟踪控制部件中的光标 5 - 35
　　　　确定和更改控制部件设定状态 5 - 37
　　　　滚动文稿 5 - 43
　　　　为响应滚动杆中的事件进行滚动 5 - 53
　　　　为响应滚动箭头和灰度面积中的事件进行滚动 5 - 57
　　　　在窗口内绘制滚动后的文稿 5 - 62
　　移动滚动杆和调整滚动杆大小 5 - 65
　　定义您自己的控制部件定义函数 5 - 71
控制管理器参考资料 5 - 72
　　数据结构 5 - 72
　　　　控制记录 5 - 73
　　　　辅助控制记录 5 - 76

弹出式菜单专用数据记录 5 - 77
控制部件色彩表记录 5 - 77
控制管理器例程 5 - 80
创建控制部件 5 - 81
绘制控制部件 5 - 85
处理控制部件中的鼠标事件 5 - 88
更改控制设定值和显示 5 - 93
确定控制部件的值 5 - 102
取消控制部件 5 - 108
应用程序定义的例程 5 - 109
定义您自己的控制部件定义函数 5 - 109
定义您自己的动作过程 5 - 115
资源 5 - 117
控制资源 5 - 118
控制部件色彩表资源 5 - 121
控制部件定义函数 5 - 123
控制管理器总结 5 - 124
Pascal 小结 5 - 124
常数 5 - 124
数据类型 5 - 126
控制管理器例程 5 - 127
应用程序定义的例程 5 - 129
C 小结 5 - 129
常数 5 - 129
数据类型 5 - 131
控制管理器例程 5 - 132
应用程序定义的例程 5 - 134
汇编语言总结 5 - 134
数据结构 5 - 134
全局变量 5 - 135

第五章

控制管理器

本章介绍了您的应用程序如何使用控制管理器来创建和管理控制部件。控制部件是那些用户可以用鼠标来操纵的屏幕上的对象。通过操纵这些控制部件，用户可以采取直接动作或者更改设定状态，以便修改将来的动作。例如，滚动杆控制部件允许用户立即更改您的应用程序所显示的文稿部分，而可控制波特速率的弹出式菜单控制部件可允许用户更改您的应用程序处理后继数据的传送速率。

阅读本章可以了解到如何以及何时实现这些控制部件。实际上，所有的应用程序都需要实现控制部件，至少要有用于文稿窗口的滚动杆。您可以使用控制管理器的例程、资源、和数据结构来实现您的应用程序文稿窗口中的滚动杆。

其它标准的Macintosh控制部件包括按钮、复选框、选项按钮、和弹出式菜单。您也可以使用控制管理器来创建和管理这些控制部件。除此之外，您可以使用对话管理器在警告栏和对话框中更为方便地实现这些控制部件。(您通常可以使用警告栏来警告用户发生了异常情况，也可以使用对话框请求用户提供执行命令所需要的信息。) 在本书的"对话管理器"一章中详尽介绍了如何在警告和对话框中实现控制部件。但是，在某些情况下－例如当您需要实现非常复杂的对话框时－您可能希望使用控制管理器例程来直接管理这些类型的控制部件；请就此问题阅读本章中的有关说明。

对于图形或文字类信息的滚动列表（类似于当用户从文件菜单中选择"新文稿"命令后，系统软件显示出的文件列表），您的应用程序可以使用列表管理器来实现滚动杆。请参阅《Inside Macintosh工具箱扩展》*(Inside Macintosh : more Macintosh Toolbox)* 的"列表管理器"一章中的详尽说明。

控制管理器提供的例程可在控制部件和屏幕刷新控制部件中自动地处理由用户生成的鼠标事件，以响应更新事件。有关事件和事件处理的详细说明，请参阅本书的"事件处理"一章。

您一般是使用控制资源（即'CNTL'类型的资源）来指定一个控制部件的类型、大小、位置和其它属性。请参阅本书的"Macintosh工具箱引论"一章中关于资源的简要说明；关于资源管理器及其例程的详尽说明请参阅《Inside Macintosh工具箱扩展》*(Inside Macintosh : more Macintosh Toolbox)* 的"资源管理器"一章。

您所创建的每个控制部件必须与一个特定的窗口相关联。用于一个窗口的所有控制部件均存储在由该窗口的窗口记录引用的控制部件列表中。请参阅本书的"窗口管理器"一章中关于窗口的简要说明。(当您使用对话管理器来实现控制部件时，对话管理器将该控制部件分别与对话框或警告栏相关联，"对话管理器"一章对此作了介绍)。

本章对控制部件的使用作了介绍，然后讨论您应如何：

- 创建和显示控制部件
- 确定控制部件中是否有鼠标按下事件
- 响应控制部件中的鼠标按下事件
- 更改控制部件中的设定状态
- 使用滚动杆来移动窗口中的文稿
- 移动和调整用于窗口的控制部件
- 定义您自己的控制部件定义函数来创建非标准的控制部件

控制部件介绍

控制管理器提供了若干标准的控制部件。图5－1是这些标准控制部件：按钮、复选框、选项按钮、弹出式菜单和滚动杆。您也可以设计和实现定义的控制部件。

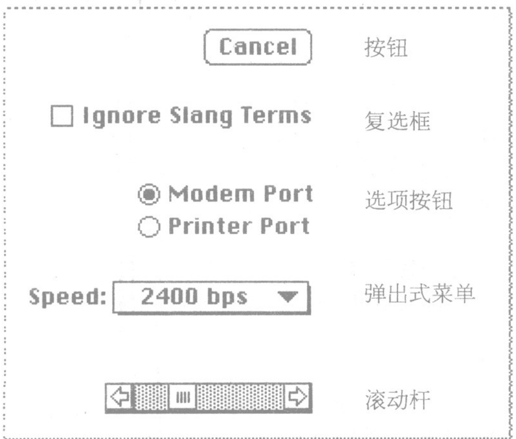


图5－1　控制管理器提供的标准控制部件

按钮、复选框和选项按钮是最简单的控制部件。它仅包含一个标题和一个轮廓线形状，并且仅响应鼠标单击。弹出式菜单稍复杂一些，此控制部件具有一个与其标题相连的菜单，当用户将拖拽光标经过该菜单时，它必须做出响应。由于滚动杆包含了行为各不相同的几个部分，所以它是最为复杂的标准控制部件。尽管一个滚动杆分为若干部分，但它仍然只是一个控制部件。

控制管理器可用各种彩色显示这些控制部件，这些美观一致的彩色适用于包括从黑白显示器到8位彩色显示器的所有显示器。若要确保各种应用程序的一致性，您一般不应更改系统设定的控制部件彩色，尽管控制管理器实际上允许您利用SetControlColor过程（在第5 － 101页介绍）或控制部件色彩表资源（在第5 － 121页介绍）这样做。

下面的几节介绍了标准控制部件和普通自定控制部件。

按钮

按钮是出现在屏幕上的圆角矩形，矩形的中央有一个标题。当用户点按一个按钮时，您的应用程序就应执行该按钮标题所示的动作。一般来说，按钮允许用户即刻执行动作－例如，完成由对话框定义的操作或者在警告栏中确认一个错误消息。

您的按钮应大到足以包含它们的标题。在显示按钮的每个窗口或对话框中，您应该指定一个按钮作为系统设定按钮，方法是围绕按钮绘制一条粗黑的轮廓线（如图5 － 2所示）。对于涉及Enter和Return键的键按下事件，您的应用程序应视其为用户点按了系统设定按钮而作出响应。（在您的警告栏中，对话管理器自动地为系统设定按钮加轮廓线；您必须在您的对话框中为系统设定按钮加轮廓线。）



图5 － 2　一个系统设定按钮

您一般可以使用警告栏和对话框中的按钮。请参阅“对话管理器”一章，该章进一步介绍了显示按钮的位置、其标题的内容是什么，如何响应涉及它们的事件，以及如何围绕它们绘制轮廓线。

复选框

复选框可以提供交替选择。您一般可在对话框中使用复选框，以便用户能够指定完成一条命令所需要的信息。复选框的操作像切换开关一样打开或关闭一个设定状态。使用复选框可以表示必须是打开或者关闭的一个或多个选项。复选框的外观像一个小方格，旁边有一个标题；利用控制管理器的过程SetControlValue可将一个选择标志“X”置于框中，当用户点按打开时就可以选定它，该过程也可删除选择标志“X”，当用户点按关闭时就可以取消选定。图5 － 3是一个选定的复选框。



图5 － 3　一个选定的复选框

当您设计一个对话框时，您可以使其中包含任意数量的复选框，也可以仅包含一个复选框。复选框之间是相互无关的，尽管有时它们提供的选项是相关的。在一个对话框内，最好是将相关的一组复选框组成一个分组集，并在不同的分组之间提供某种直观的分界线。

每个复选框有一个标题，赋予选项一个明确的标题是非常困难的。标题应该能够反映出两种明显的对立状态。例如，在Finder的Info窗口中，一个复选框提供的选项可以锁定一个文件。该复选框的标题仅是“锁定”。当该选项处于关闭状态时，其对立状态显然是“解除锁定”。

如果您未能设计出可明确表示对立状态的复选框标题，您最好是使用两个选项按钮。利用两个选项按钮，您就可以使用两个标题来区别两种状态。

在对话框中经常使用复选框设定或修改未来的动作，而不是指定立即发生的动作。请参阅本书的“对话管理器”一章中关于如何以及在哪里显示对话框中的复选框的详细讨论。

选项按钮

像复选框一样，**选项按钮**可以保持和显示一个打开或关闭设定。您可以将选项按钮组成一个分组，使用户能够在若干交替选项中进行选择－这些选项一般是在对话框中。选项按钮的形状是小圆圈；当用户单击一个选项按钮打开它时，系统使用控制管理器的过程`SetControlValue`将一个小黑点填充到选项按钮中。用户每次仅能够将一个选项按钮设定为有效。换言之，选项按钮之内是相互排它的。但是，控制管理器是不能确定您的选项按钮是如何分成分组的；因此，当用户打开一个选项按钮时，由您的应用程序调用`SetControlValue`将该分组中的其它项关闭。

一组选项按钮一般有2至7个项目；通常每组至少有两个选项按钮。每组按钮必须有一个标签，以标识该分组提供的选择种类。此外，每个按钮必须有一个标题，以标识该选项按钮的作用。此标题可以是几个单词或是一个短语。一组选项按钮完全是非动态的即其内容不可能根据其上下文变化。(如果您需要显示的项目超过七个，或者项目是随上下文的变化而变化的，您就应该使用弹出式菜单而不是选项按钮。)

选项按钮所代表的选择是相关的，但不一定对立。例如，一对选项按钮可以使用户能够在使用调制解调器端口或打印机端口之间进行选择，如第5－4页图5－1所示。如果每次看到的选项按钮超过一组，您就需要在每组之间划分界线。例如，您可以在一组选项按钮的周围绘制一条虚线，以区分对话框中的其它元素。

弹出式菜单

在本书“菜单管理器”一章中介绍的弹出式菜单使用户能够从选择列表中方便地进行选择，而不必将光标移动到菜单栏上。作为选项按钮分组的一种替代方法，弹出式菜

单尤其适用于指定含有5个或5个以上设定或值的分组，或者其设定或值需要更改的分组。像一组选项按钮中的项目一样，弹出式菜单中的项目是相互排它的－即菜单中每次只能有一个选择是有效的。第5－12页的图5－8介绍了用户在弹出式菜单中选定的选择。

请不要使用弹出式菜单向用户提供命令。弹出式菜单不应列出动作（即动词），而是应该列出属性（即形容词）或设定状态，使用户可以从中选择一个选项。

滚动杆

滚动杆可以在文稿的窗口中更改用户能够显示的文稿部分。滚动杆是一个浅灰色的矩形，两端均带有滚动箭头。滚动杆中是一个称作**滚动块**的方格。滚动杆的其余部分称为**灰度面积**。窗口可以带有一个水平滚动杆、一个垂直滚动杆、或者二者皆有。垂直滚动杆位于窗口的右侧。水平滚动杆位于窗口的底部。图5－4表示一个滚动杆的各部分。

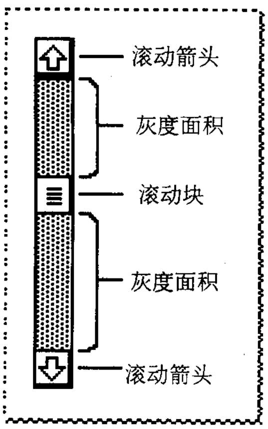


图5－4　　垂直滚动杆

如果用户拖拽滚动块、单击一个滚动箭头、或单击灰度面积的任意处，您的应用程序就相应地“移动”文稿；请使用控制管理器例程适当地移动滚动块。图5－5的图示，以及下面几节的解释，说明了一个滚动杆的若干键行为。

滚动杆以一个尺寸表示全部文稿，即从文稿的顶部至底部或者从右边至左边。滚动块可显示出文稿的可视部分在全文中的相对位置。
之间，那么用户便会看到全部文稿的中间部分。每当您的应用程序调整窗口的大小或滚动文稿（不是为响应用户拖拽滚动块）时，请使用SetControlValue或SetControlMaximum过程来移动滚动块。

在用户拖拽滚动块之后，控制管理器便在新的位置重新绘制滚动块。然后您就可使用GetControlValue函数来确定滚动块的位置，并且显示文稿的相应部分。用户只要拖拽滚动块，就可以快速地来回移动文稿。例如，若要看文稿的开始部分，用户就将滚动块拖拽到滚动杆的顶部。您的应用程序便可将文稿滚动到其顶部。

在滚动杆的两端是**滚动箭头**，用来表示移动文稿的方向。例如，当用户单击顶部的滚动箭头时，您的应用程序需要向文稿的开始部分的方向移动。这样，该文稿向下移动，好像是向开始部分的反方向移动。通过点按滚动箭头，用户就告诉了您的应用程序，“请沿此方向把隐藏的文稿显示出来。”

您的应用程序使用SetControlValue过程，按照被用户单击的箭头的方向移动滚动块。这样，滚动块就可以继续代表文稿的可视部分相对于全文的大概位置。例如，当用户单击顶部的滚动箭头时，您可将文稿向下移动以显示文稿顶部的更多内容，并且将滚动块向上移动，如图5－5所示。

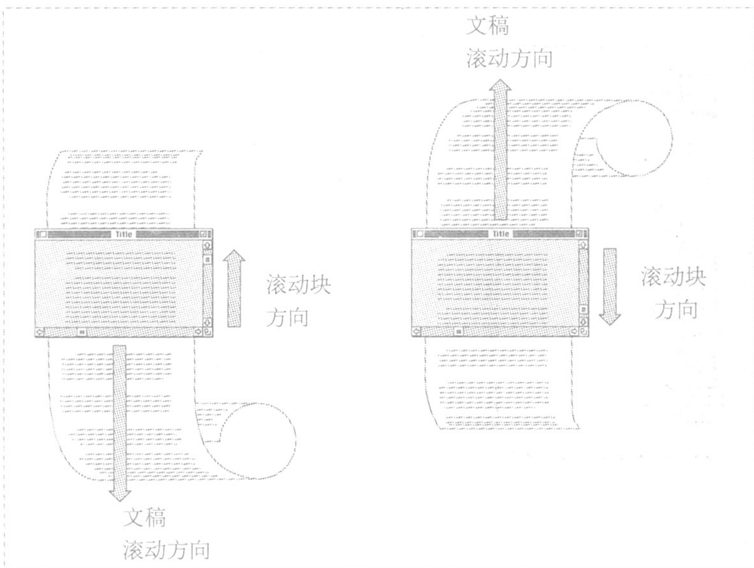


图5－5　使用滚动块和滚动箭头

每单击一次滚动箭头就应该使文稿向所选择的方向移动一个单位的距离。您的应用程序可确定一个单位等于多少。例如，在字处理程序中，每次单击箭头应该移动一行文字。在电子表格中，根据箭头的方向，应该移动一行或一列。若要确保稳定的滚动效果，一般最好在一个文稿中指定相同大小的单位。当光标位于滚动箭头时如果用户按住鼠标按钮，您的应用程序就应将文稿向该方向连续滚动，直到用户释放鼠标按钮，或者您的应用程序无法继续滚动为止。

不包括滚动块和滚动箭头，滚动杆中的其余区域称为灰度面积。当用户单击滚动杆的灰度面积时，您的应用程序移动的文稿显示区域应该是整个窗口的显示区域减去一个滚动单位。例如，如果窗口显示15行文字，当用户单击滚动块下面的灰度面积时，您的应用程序应该将文稿向上移动14行，这样就使先前屏幕显示的最后一行出现在新的屏幕显示的顶部。(此保留行可以帮助用户看到新显示的资料在上下文中的关系。)您还必须将滚动块沿该方向移动适当的距离。例如，当用户单击滚动块下面的灰度面积时，应把要显示的文稿向文稿底部方向移动一个窗口，并且使用SetControlValue相应地移动滚动块。

当您的应用程序滚动一个文稿时－例如当用户操纵滚动杆时－您的应用程序必须相对窗口的坐标空向移动该文稿的坐标空间。您的应用程序可使用滚动块来表示所显示的文稿顶部与文稿其余部分的相对位置。

例如，一个文字窗口中包含有15行文字，用户从文稿的顶部滚动了30行，那么滚动块就应该设定为数值30。在窗口中便可显示第30和第45行之间的所有行，如的图5－6所示。滚动块通常可表示出文稿的开头与文稿显示部分的顶部之间的位移。

若要避免用户在滚动时越过文稿的边缘，从而看到空白的窗口，对于垂直滚动杆，您应该使文稿滚动的长度不超过文稿的长度减去窗口的高度，这里谈及的窗口高度不包括用于窗口底部边缘水平滚动杆15像素深的区域。同样，对于水平滚动杆，您应该使文稿滚动的宽度不超过文稿宽度减去窗口宽度，这里的窗口宽度同样不包括用于窗口右侧边缘垂直滚动杆15像素宽的区域。

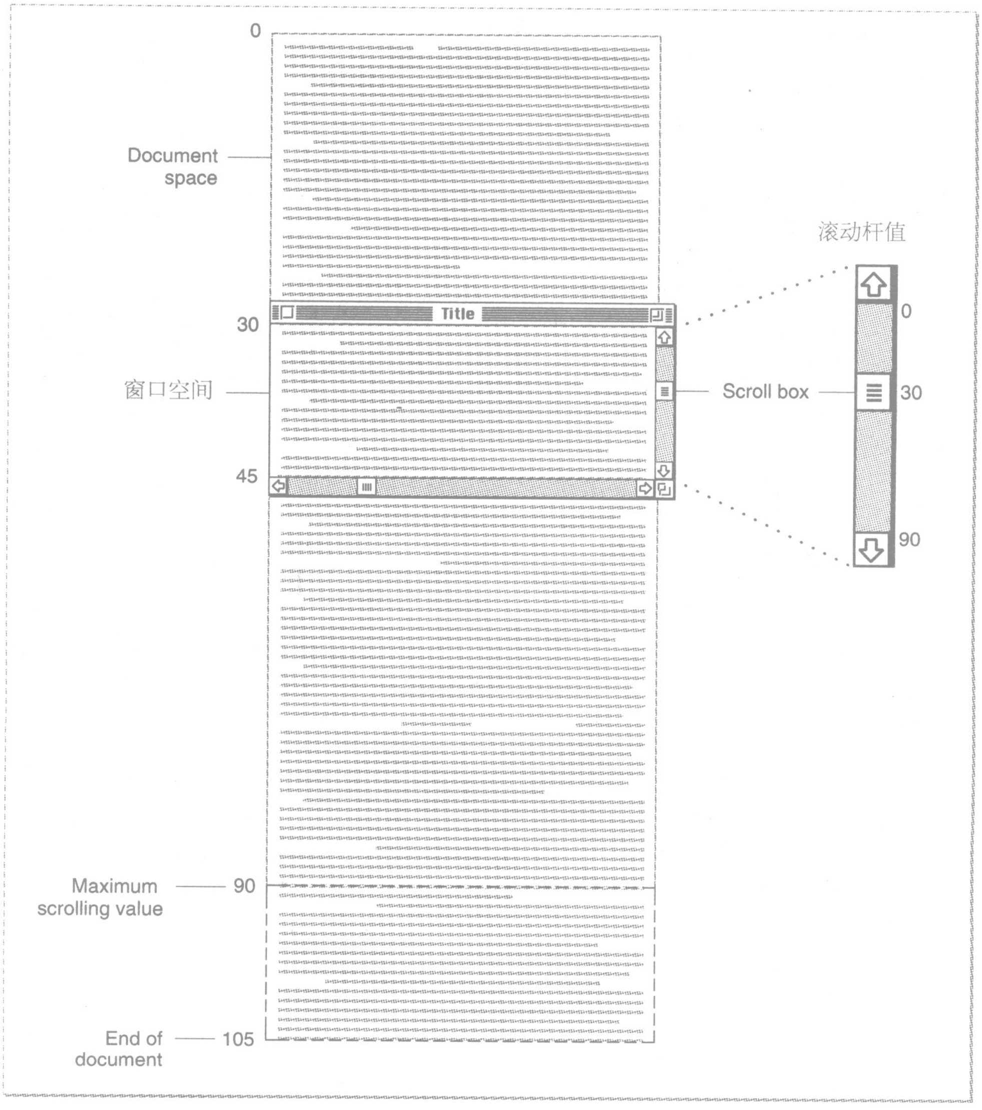


图 5 － 6　文稿与窗口的空间关系，以及用滚动杆表示的方法

例如，图 5 － 6 所示文稿的长度是 105 行。应用程序所滚动的文稿不应超过 90 行，以便当用户滚动到文稿结尾时，至少有 15 行将填充于窗口中，因为用户已经滚动到第 30 行文稿（最多可滚动到 90 行），所以滚动块出现在滚动杆三分之一位置上。

在 5 － 43 页的“滚动文稿”中详尽介绍了如何在窗口中滚动文稿。

其它控制部件

如果除控制管理器提供的标准控制部件外您还需要其它控制部件，您可以设计并执行您自己的控制部件。通常您只需要执行滑块或拨号类型的控制部件。滑块和拨号（仅在外观上不同）类似于滚动杆，因为它们是用图形表示用户可以设定的数值范围。使用一个指示符（如滑动开关或拨号针）可以表示该控制部件的当前设定状态，用户利用它可以设定一个值。（对于滚动杆来说，滚动块就是指示符。）

如果您希望显示一个不由用户直接控制的值（例如，磁盘上的空闲空间量），您就应该使用状态条或其它类型的图形，而不使用滑块或拨号。

图5－7是几个自定的控制部件，它们的用途是设定扬声器的音量、灰度饱合级、以及某一表示中一次滑动的相对位置。在此图中，要确保使用有意义的标签来表示您的控制部件中指示符的范围和方向。

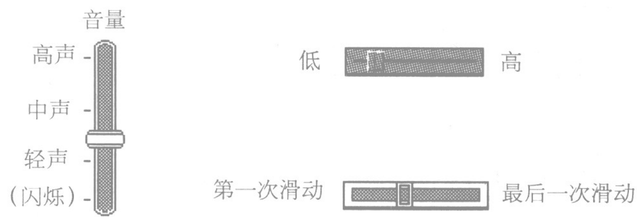


图5－7　自定的滑块控制部件

滚动杆是代表一个窗口全部内容的一个滑块，用户可使用滚动块移动到该项内容的特定位置。请不要使用滚动杆代表任何其它概念（例如更改一个设定）。否则，这种与Macintosh 界面的不一致会使用户感到混乱。

有效和无效控制部件

您可以使一个控制部件有效或无效。图5－8说明在用户将鼠标移动到一个有效控制部件并按下鼠标按钮时 `TrackControl` 函数（用来在控制部件中响应鼠标按下事件）如何作出一个可视的反馈。具体地说，`TrackControl` 可通过以下方式对有效控制部件中的鼠标事件作出响应有:

- 在反色视频中显示按钮
- 用粗线绘制复选框和选项按钮
- 醒目显示弹出式菜单标题并显示弹出式菜单中的命令项目
- 醒目显示滚动箭头
- 当用户拖拽滚动块时移动其轮廓线

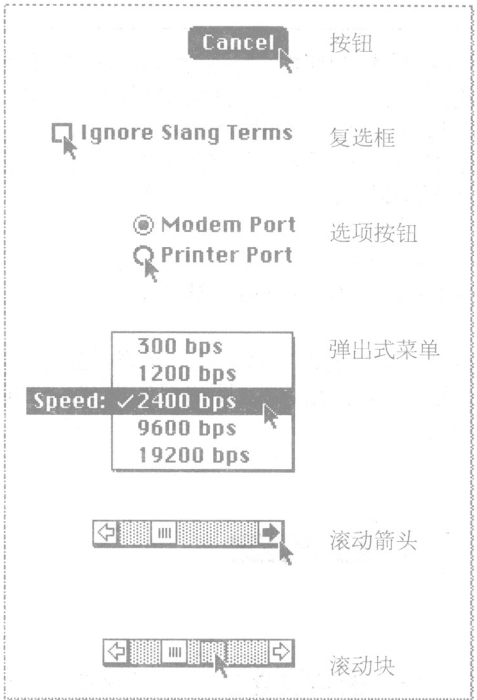


图 5－8　对于用户选定的有效控制部件所作的可视反馈

随后，您的应用程序应该相应地响应涉及有效控制部件的鼠标事件。大多数情况下，您的应用程序在执行任何动作之前须等到用户释放鼠标按钮；当光标位于一个控制部件时，只要用户按住鼠标按钮，通常您就可令 TrackControl 来反应对鼠标按下事件作出反应，然后，当光标位于有效控制部件时，一旦用户释放鼠标按钮，TrackControl 就可通知您的应用程序。

一旦用户释放鼠标按钮，您的应用程序：

- 当光标位于有效按钮时，应执行按钮标题标志的任务
- 当光标位于一个有效复选框上时，应切换复选框的值（然后控制管理器相应地绘制或删除该检查标记。）
- 当光标位于一个有效选项按钮上时，应打开该选项按钮，并关闭该分组中所有其它选项按钮
- 当光标位于一个有效的弹出式菜单时，应使用由用户选择的新的设定
- 当光标位于一个有效滚动杆的滚动箭头或灰度面积上时，应显示滚动箭头方向所指的更多的文稿部分，并相应地移动滚动块

■ 当光标位于滚动块上时，应确定用户拖拽滚动块的位置，然后显示相应部分的文稿有时，您的应用程序甚至在用户释放鼠标按钮之前就要对其响应——即您的应用程序应该采取某些连续性动作有时，您的应用程序甚至在用户释放鼠标按钮之前就要对其响应——即当光标在一个有效控制部件上时，用户按住鼠标按钮，您的应用程序应该采取某些连续性动作。最典型的情况是，如果用户把光标移至滚动杆或灰度面积，然后按住鼠标按钮，您的应用程序应该连续滚动文稿，直至用户释放鼠标按钮，或者用户已不能继续向前滚动。若要执行这一类的动作，您应该定义一个**动作过程**，并对 TrackControl 指定它；只要用户按住鼠标按钮，TrackControl 就调用您的应用程序动作过程。

只要您的应用程序不再适合响应控制部件中发生的一个鼠标按下事件，您就应该停用该控制部件。一个**无效控制**是用户不能使用的控制部件，因为它在当前的上下文中没有意义，或没有效应（例如在空窗口中的滚动杆）。控制管理器继续显示无效控制部件，使它们仍然可见，但是显示的方式向用户表明了它的状态。如图 5-9 所示，控制管理器使无效的按钮、复选框、选项按钮和弹出式菜单的彩色变得暗淡，它还使灰度面积的彩色变淡，并从无效滚动杆中删除滚动块。

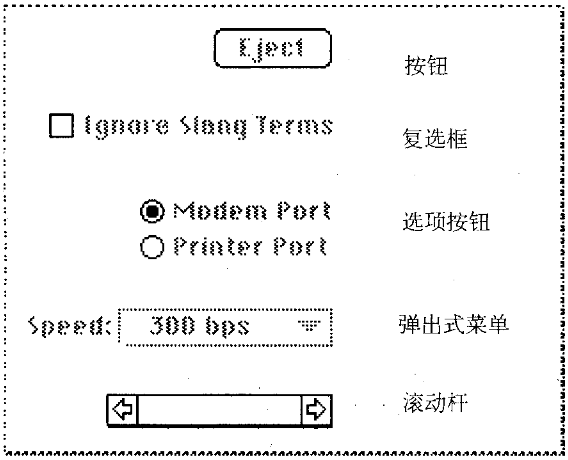


图 5-9 无效控制部件

您可以用 HiliteControl 过程令任何控制部件无效，然后再令其有效。除了滚动杆以外（您应该用 HideControl 过程隐藏它），您应该用 HiliteControl 令非当前活动窗口的所有控制部件无效。通常，您在对话框中使用除了滚动杆以外的所有其它控制部件。有关如何使按钮、选项按钮、 让它们再变成有效的讨论，请参阅本书中“对话管理器”一章。

当文稿比您显示它的窗口小时，可以使滚动杆无效。若要使滚动杆无效，您通常是用 SetControlMaximum 过程使滚动杆的最大值等于它的最小值，这时，控制管理器自动使滚动杆无效。若要再使滚动杆有效，您通常是使它的最大值比最小值大。

控制部件定义函数

控制部件定义函数决定了控制部件的外观和行为。每当各种控制管理器例程需要执行某些和控制有关的动作，例如在屏幕上绘制控制部件，它们就调用控制部件定义函数。

控制部件定义函数是作为 'CDEF' 类型的资源储存的。系统文件包括 3 种**标准控制部件定义函数**，分别以资源标识符 0、1 和 63 储存。资源标识符 为 0 的 'CDEF' 资源定义了按钮、复选框和选项按钮的外观及行为；资源标识符为 1 的 'CDEF' 资源定义了滚动杆的外观及行为；资源标识符为 63 的 'CDEF' 资源定义了弹出式菜单的外观和行为。(如果您要定义非标准的控制部件，就必须为它们编写控制部件定义函数，5-109 页的"定义您的控制部件定义函数"一节对此作了说明。)

正如窗口定义函数可以说明同一基本窗口的变形一样，控制部件定义函数可以使用**变量代码**说明同一基本控制部件的变形。您指定一个具有控制部件定义标识符的特殊控制部件。**控制部件定义标识符**是一个整数，它的高端 12 位是控制部件定义函数的资源标识符，低端 4 位是一个变量代码。对于给定的资源标识符和变量代码，控制部件定义标识符可以 按下面的公式得出：

控制部件定义标识符＝16*（'CDEF' 资源标识符）＋变量代码

例如，按钮、复选框和选项按钮都使用资源标识符为 0 的标准控制部件定义函数；由于它们的变量代码分别为 0、1 和 2，它们各自的控制部件定义标识符也就分别为 0、1 和2。

您可以使用下面的常数，定义标准控制部件定义函数提供的控制部件：

常数	**控制部件定义标识符**	**控制部件**
pushButProc	0	按钮
checkBoxProc	1	复选框
radioButProc	2	选项按钮
scrollBarProc	16	滚动杆
popupMenuProc	1008	弹出式菜单

滚动杆的控制部件定义函数能够根据您创建控制部件时指定的矩形，算出滚动杆是垂直的还是水平的。

关于控制管理器

您可以使用控制管理器执行如下任务：

- 创建和处理控制部件
- 显示、更新及隐藏控制部件

- 更改控制部件的大小、位置和外观
- 监视用户操作控制部件并作出响应
- 决定及更改控制部件的设定值和其它属性

您的应用程序通过调用适当的控制管理器例程执行这些动作。尽管实际的操作由控制管理器执行，但是何时、何地及如何执行这些动作是由您来决定的。

使用控制管理器

一般情况下，若要实现一种控制部件，您需要完成如下的工作:

- 使用控制资源（即，'CNTL' 类型的资源）说明控制部件
- 创建并显示控制部件
- 确定当光标在控制部件上时，用户单击鼠标按钮或者按住鼠标按钮的时间
- 对涉及控制部件的事件进行适当的响应——例如，当用户操纵滚动杆时，显示文稿的不同部分
- 对包括控制部件的窗口中的其它事件进行适当的响应——例如，当用户调整窗口大小时，移动并调整滚动杆的大小，或者当用户使另一个窗口变成当前活动窗口时，隐藏原来当前活动窗口的滚动杆

在本章的其余部分将对这些任务作更详细的说明。

在使用控制管理器之前，您必须使用InitGraf、Initfonts和InitWindows 过程，按照 QuickDraw、字体管理器和窗口管理器的次序逐个进行初始化。[有关InitGraf和Initfonts的信息，请参阅《Inside Macintosh绘图》*(Inside Macintosh: Imaging)* 一书；有关InitWindows的信息，请参阅本书的“窗口管理器”一章。]

创建和显示控制部件

若要在您的应用程序的一个窗口中创建控制部件，应使用GetNewControl 或NewControl函数。通常您应该使用GetNewControl，它从资源文件的控制资源（即 'CNTL' 资源）取得信息。控制资源和窗口资源一样，为了修改方便，它从您的应用程序代码中分离出说明性的信息，这在转换成其它语言时特别方便。本节的其余部分说明如何使用GetNewControl。您也可以不使用控制资源，而是使用NewControl函数，并在单个参数中传送给它说明性的信息，但是通常不推荐这样做。NewControl函数在 5-82 页中说明。

当使用GetNewControl时，您传送给它控制资源的资源标识符，以及窗口的一个指针。然后，这个函数从控制资源的信息中，创建一个ControlRecord类型的数据结构（称为控制记录），再把控制记录添加至窗口的控制列表，并返回控制部件的一个间接指针作为函数结果。（当在其它大多数控制管理器例程中引用控制部件时，您应该使用它的间接指针。当您为一个窗口创建滚动杆或弹出式菜单时，应该在您的应用程序自己的一个数据结构中储存它们的间接指针，以便后来参考。）

当您在控制资源中指定一个开始时可见的控制部件，并且您使用的是 GetNewControl 函数时，控制管理器便会通过控制部件的控制部件定义函数在窗口内绘制这个控制部件。控制管理器立即绘制该控制部件，并不使用您的窗口的标准更新机制。如果您指定控制功能不可见，您可以使用 ShowControl 过程绘制控制部件。同样，控制管理器立即绘制该控制部件，而不使用您的窗口的标准更新机制。（当然，即使当控制管理器绘制控制部件时，由于有重叠的窗口或其它对象，用户可能部分地或完全地看不清这件工作的进行。）

当您的应用程序收到包含控制部件的窗口的更新事件时，您应该在应用程序的标准窗口更新代码中使用 UpdateControls 过程，在窗口的更新区域重新绘制所有的控制部件。

附注：

当您在警告栏和对话框中使用对话管理器实现按钮、选项按钮、复选框或弹出式菜单时，对话管理器例程自动使用控制管理器例程为您创建并更新这些控制部件。如果您要在复选框和对话框中实现除了按钮、选项按钮、复选框或弹出式菜单以外的任何控制部件，或者您要在应用程序的窗口中实现任何（例如滚动杆）控制功能，您必须明确地使用 GetNewControl 或 NewControl 函数创建这些控制部件。若要更新您的窗口中的控制部件，必须使用 UpdateControls 过程。

当您使用窗口管理器过程 DisposeWindow 或 CloseWindow 删除窗口时，这两个过程都自动地删除该窗口的所有控制部件，并释放它们占据的存储器。

当您不再需要窗口中的控制部件，而仍需要保留窗口时，可以使用 DisposeControl 过程。这个过程在 5-108 页中说明，它从屏幕上删除不需要的控制部件，从窗口列表中也删除它，并释放控制记录和所有其它有关的数据结构占据的存储器。您可以使用 KillControl 过程一次性处理窗口的所有控制部件。这个过程也在 5-108 页中说明。

下一节“创建按钮、复选框或选项按钮”对控制资源进行了广泛的讨论，并对于在您的应用程序的窗口中使用控制资源指定按钮、复选框和选项按钮作了详细的说明。以后的两节“创建滚动杆”（5-21 页）和“创建弹出式菜单”（5-25 页），说明了控制资源中不同于按钮、复选框和选项按钮的那些元素。5-29 页的“更新控制部件”给出了一个实例，它说明了如何在您的窗口更新代码中使用 UpdateControls 过程。

附注:

为了使控制管理器在窗口中正确绘制控制部件，窗口左上角局部坐标必须是(0, 0)。如果您使用 QuickDraw 的过程 SetOrigin 改变了窗口的局部坐标系，必须要在绘制它的任何控制部件前改回去，使它的左上角坐标仍然为 (0, 0)。由于许多控制管理器例程能够重新绘制控制部件（至少有这个可能性），最安全的办法是，改变了窗口的局部坐标系后，在调用任何控制管理器之前改回去。

创建按钮、复选框或选项按钮

图 5-10 是在 noGrowDocProc 类型的窗口中放置一个按钮的简单的实例，通常您使用 noGrowDocProc 创建一个不定模式对话框。您一般使用对话管理器创建对话框和它们的按钮，但是您也可以使用窗口管理器和控制管理器。本书的“对话管理器”一章解释了为什么有时使用窗口管理器和控制管理器更合适。

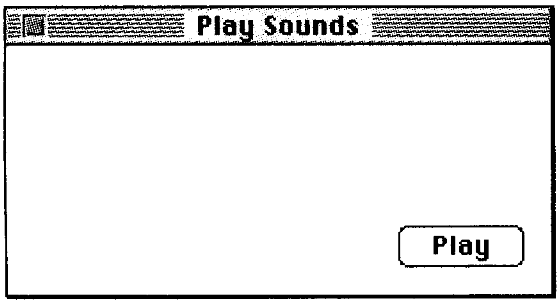


图 5-10　一个简单窗口中的按钮

程序清单 5-1 给出了一个应用程序定义的例程，MyCreatePlaySoundsWindow，它使用 GetNewControl 函数创建图 5-10 中显示的按钮。

程序清单 5-1 为窗口创建一个按钮

```
FUNCTION MyCreatePlaySoundsWindow: OSErr;
VAR
     myWindow: WindowPtr;
BEGIN
     MyCreatePlaySoundsWindow := noErr;
     myWindow := GetNewWindow(rPlaySoundsModelessWindow, NIL, POINTER(-1));
     IF myWindow <> NIL THEN
     BEGIN
          {use the window's refCon to identify this window}
          SetWRefCon(myWindow, LongInt(kMyPlaySoundsWindow));
```

```
        SetPort(myWindow);
        gMyPlayButtonCtlHandle := GetNewControl(rPlayButton, myWindow);
        IF (gMyPlayButtonCtlHandle = NIL) THEN
            MyCreatePlaySoundsWindow := kControlErr;
    END
    ELSE
        MyCreatePlaySoundsWindow := kNoSoundWindow;
END;
```

MyCreatePlaySoundsWindow 例程一开始就使用窗口管理器函数 GetNew-Window 创建一个窗口；这个窗口的指针被传送给 GetNewControl。注意，本书的“对话管理器”一章中已经说明过，创建不定模式对话框更容易的方法是使用对话管理器的函数 GetNewDialog 并在项目列表（ 'DITL' ）资源中指定它的控制部件。

MyCreatePlaySoundsWindow例程为控制资源的资源标识符 定义了rPlayButton 常数，并把它传送给 GetNewControl 函数。程序清单 5-2 给出了 Rez 输入格式下的这个控制资源。

程序清单 5-2 控制资源的 Rez 输入

```
resource 'CNTL' (rPlayButton, preload, purgeable) {
    {87, 187, 107, 247},            /*rectangle*/
    0,                              /*initial setting*/
    visible,                        /*make control visible*/
    1,                              /*maximum setting*/
    0,                              /*minimum setting*/
    pushButProc,                    /*control definition ID*/
    0,                              /*reference value*/
    "Play"                          /*title*/
};
```

您在控制资源中为按钮、复选框、选项按钮或滚动杆提供如下信息:

- 以窗口局部坐标指定的一个矩形，它决定了控制部件的大小和位置
- 控制部件的初始设定值
- 一个常数（可以是 visible 也可以是 invisible)，它指定了是否应该立即在屏幕上绘制控制部件
- 控制部件的最大设定值
- 控制部件的最小设定值
- 控制部件定义标识符
- 一个参考值，您的应用程序可以将它用于任何目的
- 控制部件的标题；对于滚动杆，它是一个空字符串5-25 页的“创建弹出式菜单”一节中将会说明，您在弹出式菜单的控制资源中提供的常数，和您为按钮、复选框、选项按钮和滚动杆提供的是不同的。

绘制的按钮必须和矩形准确配合。若要允许容纳系统中最高的字符，矩形的顶部和底部的坐标差至少是 20 点。程序清单 5-2 使用的矩形坐标是（87，187，107，247），它说明了控制部件在窗口内的大小和位置。必须记住，如果窗口的左上角坐标不是（0，0），控制管理器就不能正确地绘制控制部件。

在程序清单 5-2 中，按钮的初始和最小设定值是 0，最大设定值是 1。在按钮、复选框和选项按钮的控制资源中，应提供以下的值作为初始设定值:

- 按钮不保持一个设定值。所以，对于按钮，初始和最小设定值是 0，最大设定值是 1。
- 复选框和选项按钮保持一个开关设定值。如果您要控制部件初始状态为“关闭”状态，就指定初始设定值为 0。若要复选框和选项按钮的初始状态为“开”，指定初始设定值为 1。这时，控制管理器在复选框中放置一个选择标志“X”，在选项按钮中放置一个黑点。

由于在这个实例中指定了 visible 参数值，控制部件在它的窗口中立即被绘制。如果您使用 invisible 参数值，只有在您调用 ShowControl 过程时，才绘制控制部件。如果您要使可见的控制部件变成不可见，应该使用 HideControl 过程。

在程序清单 5-2 中，按钮的最大设定值是 1，您在自己的控制资源中也应该指定按钮、复选框和选项按钮的最大设定值是 1。程序清单 5-2 中的最小设定值是 0，您在自己的控制资源中也应该指定按钮、复选框和选项按钮的最小设定值是 0。

在程序清单 5-2 中，pushButProc 常数是用来指定控制部件定义标识符的。应使用 checkBoxProc 常数指定复选框的控制部件定义标识符，使用 radioButProc 常数指定选项按钮的控制部件定义标识符。

程序清单 5-2 指定了一个参考值 0。您的应用程序可以将它用于任何目的（除了您把 popupUseAddResMenu 变量代码添加至 popupMenuProc 控制部件定义函数的情况，5-25 页的“创建弹出式菜单”一节中对此作了说明）。

最后，字符串“Play”被指定为控制部件的标题。按钮、复选框和选项按钮需要一个标题向用户说明自己的用途。（本书的“对话管理器”一章为按钮选择适当的标题提供了广泛的准则。）

当指定标题时，应保证它在控制矩形内，否则，控制管理器将截断该标题。例如，在 Roman 语系中，它在右边截断复选框和选项按钮的标题；它把按钮的标题居中，并从两边进行截断。

如果您把自己的应用程序本地化，并和系统软件的通用版本一起使用，标题可能变得更长或较短。转换的文字通常比美国英语长 50%。为了适应转换的文字，您可能需要调整控制部件的大小。

控制管理器的系统设定状态是用系统字体显示控制部件。若要方便您的应用程序的本地化，并和系统软件的通用版本一起使用，就不应该更改系统字体。不要使用较小的字体，例如 9 点 Geneva 字体；一些语系系统，比如 KanjiTalk，要求 12 点的字体。通过把所有的控制部件字体保持为系统字体，您可以为应用程序将来的本地化减轻许多负担。

控制部件标题的大写，应遵循书籍标题的字形。一般说来，单字标题首字母大写，多字标题中的名词、形容词、动词和 4 个或 4 个以上的字母的介词的首字母大写。通常 *in*、*an* 或 *and* 这些字的首字母不大写。有关大写的规则，请参阅《Apple Publication Style Guide》，它可以从 APDA 获得。

控制管理器允许按钮、复选框和选项按钮使用多行标题。当使用多行标题时，在每一行的结尾用 ASCI 字符代码 $0D（回车）结束。如果控制部件是一个按钮，每一行都是水平居中的，行之间插入和字体相应的行距。(每一行的高度等于所使用字体的上升线和下降线之间的距离加上相应于该字体的行距。应该保证矩形的总高度比行高度乘以行数要大。) 如果控制部件是复选框或选项按钮，文字对齐的方式取决于用户当前的语系系统，复选框或选项按钮的文字在它们的矩形内垂直居中。

图 5-11 是一个播放声音窗口，它有 4 个附加的控制部件：选项按钮“滴嗒”、“嘎嘎”、“哔哔”和“野外吃声”。

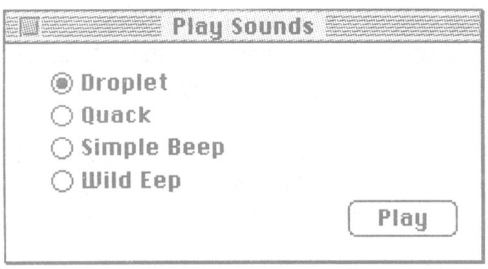


图 5-11 简单窗口中的选项按钮

这些选项按钮同时只能打开一个。程序清单 5-3 把选项按钮“滴嗒”的初始值设定为 1，所以系统设定状态是这个选项按钮处于“打开”状态。这个列表还给出了其它按钮的控制资源，它们的初始设定值都是 0，因此初始状态是关闭的。

为了放置系统字体中最高的字符，复选框或选项按钮矩形的顶部和底部坐标差至少应有 16 点。

程序清单 5-3 选项按钮控制资源的 Rez 输入

```
resource 'CNTL' (cDroplet, preload, purgeable) {
    {13, 23, 31, 142},          /*rectangle of control*/
    1,                          /*initial setting*/
    visible,                    /*make control visible*/
    1,                          /*maximum setting*/
    0,                          /*minimum setting*/
    radioButProc,               /*control definition ID*/
    0,                          /*reference value*/
    "Droplet"                   /*control title*/
};
resource 'CNTL' (cQuack, preload, purgeable) {
    {31, 23, 49, 142},          /*rectangle of control*/
    0,                          /*initial setting*/
    visible, 1, 0, radioButProc, 0, "Quack"};

resource 'CNTL' (cSimpleBeep, preload, purgeable) {
    {49, 23, 67, 142},          /*rectangle of control*/
    0,                          /*initial setting*/
    visible, 1, 0, radioButProc, 0, "Simple Beep"};

resource 'CNTL' (cWildEep, preload, purgeable) {
    {67, 23, 85, 142},          /*rectangle of control*/
    0,                          /*initial setting*/
    visible, 1, 0, radioButProc, 0, "Wild Eep"};
```

创建滚动杆

当您定义滚动杆的控制资源时，应指定常数scrollBarProc为控制部件定义标识符。通常您使滚动杆不可见，指定初始值为 0，最小值为0，并为标题提供一个空字符串。

在您创建窗口后，应使用GetNewControl函数创建您在控制资源中定义的滚动杆，并把这个滚动杆附加给窗口。可以使用MoveControl、SizeControl、SetControlMaximum和SetControlValue过程调整滚动杆的位置、大小和设定值，并使用ShowControl显示滚动杆。

在您的窗口处理代码中，应该令滚动杆最大的设定值为您允许用户滚动的最大面积。大多数应用程序允许用户拖拽尺寸格和单击缩放格改变窗口的大小，它们还允许用户在文稿中添加或删除信息。若要允许用户执行这些动作，您的应用程序需要根据文稿的当前大小及其窗口的当前大小，计算改变的最大设定值。新的文稿没有需要滚动的内容，这时应该在控制资源中把初始值 0 指派为最大值；当滚动杆的最大和最小设定值相等时，控制部件定义函数自动使滚动杆无效。以后，您的窗口处理例程应该设定并保留最大设定值。5-37 页的“决定和更改控制部件的设定值”中对此作了说明。

按照惯例，滚动杆的宽是 16 个像素，所以垂直滚动杆左边和右边的坐标差或水平滚动杆的顶部和底部的坐标差都应该是 16 点。(如果您提供的不是 16 像素的宽度，控制管理器缩放滚动杆以适合您指定的宽度。)为了使滚动箭头和滚动块有足够的空间，标准的滚动杆至少应该有 48 个像素长。

控制管理器为包容滚动杆的矩形绘制的线宽为 1 个像素。如图 5-12 所示，滚动杆的外线应该和窗口管理器为窗口框架绘制的线重叠。

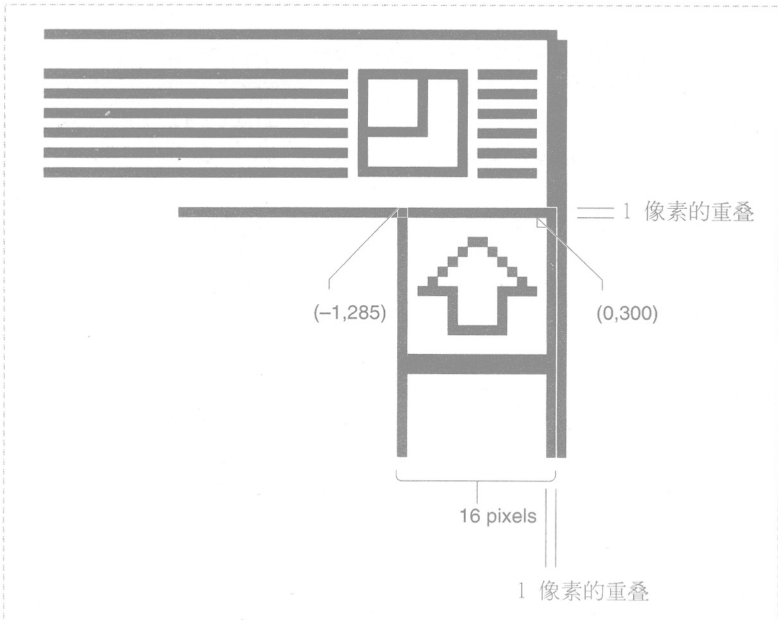


图 5-12　滚动杆如何和窗口框架重叠

若要确定一个垂直滚动杆的矩形，应执行下面的计算，并在您的控制资源中使用计算的结果。(在您的计算中不要包括标题栏的面积。)

- 顶部坐标＝在滚动杆上面的所有项目的高度和 － 1
- 左边坐标＝窗口宽度 － 15
- 底部坐标＝窗口高度 － 14
- 右边坐标＝窗口宽度 ＋ 1

若要确定一个水平滚动杆的矩形，应执行下面的计算，并在您的控制资源中使用计算的结果。

- 顶部坐标＝窗口高度 － 15
- 左边坐标＝在滚动杆左边的所有项目的宽度和 － 1
- 底部坐标＝窗口高度 － 1
- 右边坐标＝窗口宽度 － 14

如果您的应用程序不使用标准的窗口滚动杆面积（特别是靠近尺寸格的那些面积）显示信息或指定附加的控制部件，垂直滚动杆的顶部坐标是 -1，水平滚动杆的左边坐标是 -1。例如，您的应用程序可能选择在窗口的左下角一即，在窗口的水平滚动杆左边的一块小面积中显示文稿当前页的页码。有关把窗口的滚动杆面积适当用作附加项目和控制部件的讨论，请参阅《Macintosh 用户界面设计原则》*(Macintosh Human Interface Guidelines)* 一书。

当用户调整窗口大小时，滚动杆坐标位置要改变，这就像用户调整文稿窗口大小时，窗口滚动杆的最大设定值要改变一样。虽然您在控制资源中必须指定滚动栏的初始最大设定值和位置，您的应用程序还必须能够动态地更改它们。通常的做法是，当您创建窗口时，在文稿记录中储存每个滚动杆的间接指针，然后使用控制管理器例程更改控制部件设定值（5-37 页的“确定和更改控制部件设定状态”中对此将作说明）及控制部件的大小和位置（5-65 页的“移动滚动杆及调整滚动杆的大小”中对此将作说明）。

程序清单 5-4 给出了用来创建窗口的一个窗口资源（请参阅本书的“窗口管理器”一章）和创建窗口的垂直和水平滚动杆的两个控制资源。窗口的初始大小和形状以全局坐标指定，而两个滚动杆的矩形以窗口的局部坐标指定。

程序清单 5-4 窗口资源和它的滚动杆资源的 Rez 输入

```
                                /*initial window*/
resource 'WIND' (rDocWindow, preload, purgeable) {
     {64, 60, 314, 460},        /*initial rectangle for window*/
     zoomDocProc, invisible, goAway, 0x0, "untitled"
};
                                /*initial vertical scroll bar*/
resource 'CNTL' (rVScroll, preload, purgeable) {
     {-1, 385, 236, 401},       /*initial rectangle for control*/
     /*initial setting, visibility, max, min, ID, refcon, title*/
     0, invisible, 0, 0, scrollBarProc, 0, ""
};
```

```
                                /*initial horizontal scroll bar*/
resource 'CNTL' (rHScroll, preload, purgeable) {
    {235, -1, 251, 386},    /*initial rectangle for control*/
    /*initial setting, visibility, max, min, ID, refcon, title*/
    0, invisible, 0, 0, scrollBarProc, 0, ""
};
```

程序清单 5-5 给出了一个称为 DoNew 的应用程序定义的过程,它使用 GetNewWindow 和 GetNewControl 函数，按照程序清单 5-4 中的资源创建一个窗口和它的滚动杆。

程序清单 5-5 创建带有滚动杆的窗口

```
PROCEDURE DoNew (newDocument: Boolean; VAR window: WindowPtr);
VAR
    good:                           Boolean;
    windStorage:                    Ptr;
    myData:                         MyDocRecHnd;
BEGIN
    {use GetNewWindow or GetNewCWindow to create the window here}
myData := MyDocRecHnd(NewHandle(SIZEOF(MyDocRec)));{create document rec}
{test for errors along the way; if there are none, create the scroll }
    { bars and save their handles in myData}
    IF good THEN
    BEGIN   {create the vertical scroll bar and save its handle}
        myData^^.vScrollBar := GetNewControl(rVScroll, window);
        {create the horizontal scroll bar and save its handle}
        myData^^.hScrollBar := GetNewControl(rHScroll, window);
        good := (vScrollBar <> NIL) AND (hScrollBar <> NIL);
    END;
    IF good THEN
    BEGIN{adjust size, location, settings, and visibility of scroll bars}
        MyAdjustScrollBars(window, FALSE);
            {perform other initialization here}
        IF NOT newDocument THEN
            ShowWindow(window);
    END;
    {clean up here}
END; {DoNew}
```

DoNew 创建一个文稿记录，并把一个间接指针储存在它的 myData 变量中。示例应用程序 SurfWriter 使用这个文稿记录储存用户在该窗口中创建的数据，并储存 DoNew 创建的滚动杆的间接指针。以后，示例应用程序 SurfWriter 使用这些控制功能间接指针处理文稿的滚动，并当用户调整窗口大小时，移动滚动杆及调整它的大小。(有关创建这种文稿记录更详细的信息，请参阅本书的“窗口管理器”一章。)

为了创建滚动杆，DoNew 两次使用 GetNewControl，一次用于垂直滚动杆，一次用于水平滚动杆。GetNewControl 函数返回一个控制部件间接指针 DoNew 在它的文稿记录的域 vScrollBar 和 hScrollBar 中储存这些间接指针作为以后的参考资料。

因为窗口和滚动杆是不可见的，所以用户并未在屏幕上看到它们。在绘制窗口和它的滚动杆之前，DoNew 调用另一个应用程序定义的过程 MyAdjustScrollBars。这个过程又调用其它应用程序定义的例程，这些例程配合窗口移动滚动杆并调整它大小，然后计算这些控制部件的最小设定值。(5-39 页的程序清单 5-14 给出了 MyAdjustScrollbars 过程。)

在创建了窗口和它的滚动杆，并适当地调整它们的大小和位置以后，DoNew 使用窗口管理器过程 ShowWindow 显示该窗口和它的滚动杆。

创建弹出式菜单

您在控制资源中为弹出式菜单指定的值和您为其它控制部件指定的值是不同的。弹出式菜单的控制资源包括以下信息:

- 以窗口的局部坐标指定的一个矩形，它决定了弹出式标题和弹出式方框的大小和位置
- 弹出式标题和弹出式方框的对齐方式
- 一个常数 (visible 或 invisible)，它指定是否立即在屏幕上绘制控制部件
- 弹出式标题的宽度
- 说明弹出式菜单命令项目的 'MENU' 资源的资源标识符
- 控制部件定义标识符
- 一个参考值，您可以把它用于任何目的
- 控制部件的标题

图 5-13 显示了一个弹出式菜单；程序清单 5-6 给出了创建这个弹出式菜单的控制资源。(本书的“菜单管理器”一章中向您推荐了弹出式菜单的典型用途，并说明了弹出式菜单和您在菜单栏上显示的菜单间的关系。)



图 5-13 一个弹出式菜单

程序清单 5-6 弹出式菜单的控制资源的 Rez 输入

```
resource 'CNTL' (kPopUpCNTL, preload, purgeable) {
    {90, 18, 109, 198},              /*rectangle of control*/
    popupTitleLeftJust,              /*title position*/
    visible,                         /*make control visible*/
    50,                              /*pixel width of title*/
    kPopUpMenu,                      /*'MENU' resource ID*/
    popupMenuCDEFProc,               /*control definition ID*/
    0,                               /*reference value*/
    "Speed:"                         /*control title*/
};
```

程序清单 5-6 指定了一个坐标为 (90, 18, 109, 198) 的矩形。图 5-14 对弹出式菜单的这个矩形作了说明。

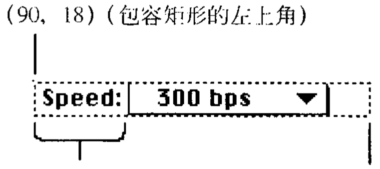


图 5-14 示例弹出式菜单的尺寸

程序清单 5-6 使用了 popupTitleLeftJust 常数指定控制部件标题的位置。下列常数（或它们的值）的任何组合，规定了绘制弹出式菜单标题的位置和方式，控制管理器根据收到的信息进行绘制。

设定值	常数	说明
$0000	popupTitleLeftJust	把标题从弹出式方框的左边开始放置
$0001	popupTitleCenterJust	标题在弹出式方框中居中
$00FF	popupTitleRightJust	把标题从弹出式方框的右边开始放置
$0100	popupTitleBold	使用黑体字形
$0200	popupTitleItalic	使用斜体字形

设定值	常数	说明
$0400	popupTitleUnderline	使用下划线字形
$0800	popupTitleOutline	使用轮廓线字形
$1000	popupTitleShadow	使用阴影字形
$2000	popupTitleCondense	使用压缩字符
$4000	popupTitleExtend	使用扩展字符
$8000	popupTitleNoStyle	使用单字形字体

如果GetNewControl成功地完成，它通过把第一个菜单命令项目的项目编号指派给控制记录的contrlValue域，设定该域的值。当用户选择一个不同的菜单命令项目时，控制管理器把contrlValue域的值更改为该项目的编号。

当您创建弹出式菜单时，您的应用程序应该为它们储存间接指针；例如，在窗口记录或对话记录的refCon域指向的记录中储存。(有关窗口记录和对话记录更详细的信息，请参阅本书的“窗口管理器”和“对话管理器”两章。）您应该按照下面的代码段所示，储存这些间接指针，使您的应用程序能够在以后响应用户的选择。

```
myData: MyDocRecHnd;
window: WindowPtr;

myData^^.popUpControlHandle := GetNewControl(kPopUpCNTL, window);
```

程序清单 5-6 指定了 50 个像素（代替最大设定值）作为控制部件标题的宽度。在控制管理器创建了控制部件后，它把弹出式菜单的控制记录中的最大值设定为弹出式菜单的项目数。图 5-14 说明了弹出式菜单的这个标题宽度。

程序清单 5-6 使用一个常数kPopUpMenu指定 'MENU' 资源的资源标识符（代替最大设定值)。(请参阅本书的“菜单管理器”一章中对 'MENU' 资源类型的说明。）在控制管理器创建了控制部件后，它指派 1 作为弹出式菜单控制记录中的最小设定值。

重要附注:

当使用2.1.1 版的ResEdit应用程序时，您在指定菜单资源和控制资源时必须使用同样的资源标识符，这两种资源一起定义了弹出式菜单。

您也可以通过把下列常数（或它们代表的的变量代码）的全部或部分添加至popupMenuProc常数，指定一个不同的控制部件定义标识符:

```
CONST     popupFixedWidth    = $0001; {use fixed-width control}
          popupUseAddResMenu= $0004; {use resource for menu items}
          popupUseWFont      = $0008; {use window font}
```

常数	说明
popupFixedWidth	使用控制部件宽度常数。如果您的应用程序指定了这个值，弹出式控制部件定义函数在水平方向不调整控制部件的大小以适应长的菜单命令项目。弹出式方框的宽度被设定为控制部件的宽度，减去应用程序创建控制部件时指定的弹出式标题的宽度。如果弹出式方框中的菜单命令项目在所提供的空间中放不下，文字就按方框宽度截断并在后面加上省略号（...）。如果您不指定这个变量代码，弹出式控制部件定义函数可能水平地调整控制功能的大小。
popupUseAddResMenu	从 'MENU' 资源以外的其它资源获得菜单命令项目。如果您的应用程序在创建弹出式菜单时指定这个值，控制部件定义函数把控制记录的 contrlRfCon 域中的这个值解释为 ResType 类型的值。控制定义函数使用菜单管理器过程 AppendResMenu 把该类型的资源添加至菜单。
popupUseWFont	使用指定窗口的字体。如果您的应用程序指定了这个值，弹出式控制部件定义函数使用包含控制部件的窗口的字体和大小绘制弹出式菜单标题，而不使用系统字体。

您的应用程序可以使用您在控制资源中指定的参考值（控制管理器把它储存在控制记录的 contrlRfCon 域中）。但是，如果您指定 popupUseAddResMenu 作为变量代码，控制管理器把控制记录的 contrlRfCon 域中的这个值强制规定为 ResType 类型，然后使用 AppendResMenu 把这种类型的项目添加至弹出式菜单。例如，如果您指定一个参考值 LongInt('FONT')，控制部件定义函数就把系统中安装的字体的一个列表增补到和弹出式菜单有关的菜单中。在创建控制部件后，您的应用程序可以按需要任意使用控制记录的 contrlRfCon 域。您可以通过调用 GetControlValue 确定当前选择的菜单命令项目。

每当重新绘制弹出式菜单时，它的控制部件定义函数就调用菜单管理器过程 CalcMenuSize。这个过程重新计算控制部件菜单的大小（为了允许在菜单中添加或删除项目）。弹出式控制部件定义函数可能还要更新弹出式菜单的宽度，使它等于弹出式标题的宽度、菜单中最长项目的宽度和向下箭头的宽度之和再加上一个为空间隔留出的余量。前面已经说过，您的应用程序可以通过对弹出式控制部件定义标识符添加变量代码 popupFixedWidth 取消以上的过程。

您不应该使用菜单管理器的函数 GetMenuHandle 取得弹出式控制部件的菜单间接指针。如果必要，您可以通过间接引用弹出式菜单控制记录的 contrlData 域获得菜单间接指针（以及菜单标识符）。控制记录的 contrlData 域是一块专用信息的间接指针。对弹出式菜单而言，这个域就是弹出式菜单专用数据记录的间接指针。5-77 页将对这一点进行说明。

更新控制部件

在收到包含控制部件（例如滚动杆）的窗口的更新事件后，您的程序应该使用 UpdateControls 过程。(窗口管理器例程，例如 SelectWindow、ShowWindow 和 BringToFront 不会自动调用 UpdateControls 显示窗口的控制部件。它们只是把适当的区域添加至窗口的更新区域。这就又生成一个更新事件。)

附注:

对话管理器自动更新您在警告栏和对话框中使用的控制部件。

当您的应用程序收到包含控制部件的窗口的一个更新事件时，应在您的窗口更新代码中使用UpdateControls过程重新绘制窗口更新区域的所有控制部件UpdateControls 的调用应该在窗口管理器的过程 BeginUpdate 使用之后，在窗口管理器的过程 EndUpdate 使用之前。

当您调用 UpdateControls 时，您传送给它的参数指定需要更新的窗口和需要更新的窗口面积。应使用窗口图形端口的可视区域指定窗口的更新区域，这个可视区域在端口的 visRgn 域中。

程序清单 5-7 给出了一个应用程序定义的例程 DoUpdate，它对一个更新事件作出响应。DoUpdate 例程调用窗口管理器的过程 BeginUpdate。然后为了重新绘制窗口的这一部分，DoUpdate 调用它自己的另一个过程 MyDrawWindow。

程序清单 5-7 响应窗口的更新事件

```
PROCEDURE DoUpdate (window: WindowPtr);
VAR
    windowType: Integer;
BEGIN
    windowType := MyGetWindowType(window);
    CASE windowType OF
    kMyDocWindow:
        BEGIN
            BeginUpdate(window);
            MyDrawWindow(window);
            EndUpdate(window);
        END;      {of updating document windows}
    {handle other window types—modeless dialogs, etc.—here}
    END;      {of windowType CASE}
END;      {of DoUpdate}
```

程序清单 5-8 说明了示例应用程序 SurfWriter 如何使用它自己的应用程序定义的例程MyDrawWindow更新窗口控制部件和窗口的其它内容。为了在窗口的更新区域只绘制需要更新的控制部件，MyDrawWindow 调用 UpdateControls。为了绘制窗口右下角的尺寸格，MyDrawWindow调用窗口管理器的过程DrawGrowIcon。最后，MyDrawWindow 重新绘制包含在用户文稿中的适当信息。因为应用程序 SurfWriter 在窗口内容的所有文字编辑中使用 TextEdit，所以程序清单 5-8 调用 TextEdit 的过程 TEUpdate。[《Inside Macintosh 文本》(*Inside Macintosh: Text*) 一书中详细说明了 TextEdit。]

程序清单 5-8 重新绘制更新区域中的控制部件

```
PROCEDURE MyDrawWindow (window: WindowPtr);
VAR
     myData: MyDocRecHnd;
BEGIN                          {draw the contents of the window}
     SetPort(window);
     myData := MyDocRecHnd(GetWRefCon(window));
     HLock(Handle(myData));
     WITH window^ DO
     BEGIN
          EraseRect(portRect);
          UpdateControls(window, visRgn);
          DrawGrowIcon(window);
          TEUpdate(portRect, myData^^.editRec);
                                   {redraw text}
     END;
     HUnLock(Handle(myData));
END;      {MyDrawWindow}
```

有关更新窗口内容的详细信息，请参阅本书的“窗口管理器”一章。

响应控制部件中的鼠标事件

控制管理器提供几个例程帮助您检测并响应涉及控制部件的鼠标事件。对于涉及控制部件的鼠标事件，您的应用程序一般要执行下列任务：

1. 在您的事件处理代码中，首先通过窗口管理器函数 FindWindow 确定发生鼠标按下事件的窗口。

2. 如果鼠标按下事件发生在应用程序的当前活动窗口的内容区域，应使用 FindControl 函数确定鼠标按下事件是否发生在有效的控制部件，如果是，发生在哪一个控制部件。

3. 应调用 TrackControl，在用户按住鼠标按钮的整个期间里处理用户和控制部件的对话。对于滚动箭头和滚动杆的灰度面积，您必须定义 TrackControl 可以使用的一个。

动作过程。这个动作过程应该使文稿在用户按住鼠标按钮的全部时间内卷动。对于弹出式菜单，您把 Pointer (-1) 传送给 TrackControl 的一个参数，以便使用弹出式控制部件定义函数中定义的动作过程。对于滚动杆中的滚动块和其它的标准控制部件，您传送NIL给 TrackControl 过程的一个参数，以便获得控制管理器对鼠标按下事件的标准响应。

4. 当 TrackControl 向您报告用户已经释放了鼠标按钮，而光标正在一个控制部件中时，您应该适当地响应。这可能要求您使用其它的控制管理器例程，例如 GetControlValue 和 SetControlValue，以便确定并更改控制部件的设定值。

以上这些和其它响应涉及控制部件的事件的例程，将在下面几节中说明。

附注：

对话管理器的过程ModalDialog对于涉及警告栏和定模式对话框的控制部件的鼠标按下事件，自动调用 FindWindow、FindControl 和 TrackControl。您可以使用对话管理器函数DialogSelect帮助您处理可移动定模式对话框和不定模式对话框中的鼠标事件。DialogSelect 自动调用 FindWindow、FindControl 和 TrackControl。

确定控制部件中的鼠标按下事件

当您的应用程序收到一个鼠标按下事件时，应使用窗口管理器函数FindWindow确定发生事件的窗口。如果用户按下鼠标按钮的同时，光标在应用程序的当前活动窗口的内容区域，应使用FindControl函数确定鼠标按下事件是否发生在有效的控制部件，如果是，发生在哪一个控制部件。

当鼠标按下事件发生在一个可见的、有效的控制部件中时，FindControl 返回该控制部件的--个间接指针，以及一个确定控制部件哪一部分的部分代码。（注意，当鼠标按下事件发生在一个不可见或无效的控制部件，或者当光标不在控制部件上时，FindControl 把控制部件间接指针设定为NIL，并返回 0 作为部分代码。）

一个简单的控制部件，例如按钮或复选框可能只有一个“部分”；更复杂的控制功能可以包含的部分数取决于控制部件如何操作的需要。滚动杆有 5 个部分：2个滚动箭头，滚动块和滚动块两边的 2 个灰度面积。5-7 页中的图 5-4 显示了滚动杆的 5 个部分。

部分代码是一个从 0 到 253 的整数，它确定控制部件的一个控制部分。为了允许具有多个控制部分的控制部件的不同部分以不同的方式响应鼠标事件，控制管理器的许多例程都接受一个部分代码作为参数或返回一个部分代码作为结果。部分代码由控制部件定义函数指派给控制部件。标准控制部件定义函数定义下列部分代码，同时列出的还有您可以用来代表部分代码的常数。

常数	部分代码	控制部分
inButton	10	按钮
inCheckBox	11	整个复选框或选项按钮
inUpButton	20	垂直滚动杆的上滚动箭头，水平滚动杆的左滚动箭头
inDownButton	21	垂直滚动杆的下滚动箭头，水平滚动杆的右滚动箭头
inPageup	22	垂直滚动杆的滚动块上面的灰度面积，水平滚动杆的滚动块左面的灰度面积
inPageDown	23	垂直滚动杆的滚动块下面的灰度面积，水平滚动杆的滚动块右面的灰度面积
inThumb	129	滚动块

弹出式控制部件定义函数不对弹出式菜单定义部分代码。您的应用程序应该在您创建弹出式菜单时，储存它们的间接指针（5-25 页的“创建弹出式菜单”一节中已有说明）。您的应用程序在响应用户在弹出式菜单中作的选择前，应该根据 FindControl 返回的间接指针检测您储存的间接指针。下一节将对这些内容进行更详细的说明。

程序清单 5-9 给出了一个应用程序定义的过程 DoMouseDown，应用程序可以调用它响应鼠标按下事件。DoMouseDown例程首先调用窗口管理器的函数FindWindow，该函数返回 2 个值：发生鼠标按下事件的窗口的间接指针，及关于事件发生位置的附加信息的一个常数。如果FindWindow返回常数inContent，鼠标按下事件就发生在应用程序一个窗口的内容区域。

程序清单 5-9 检测窗口中的鼠标按下事件

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
     part:          Integer;
     thisWindow:    WindowPtr;
BEGIN               {handle mouse-down event}
     part := FindWindow(event.where, thisWindow);
     CASE part OF
          inMenuBar:
          ;{mouse-down in menu bar, respond appropriately here}
          inContent:
          IF thisWindow <> FrontWindow THEN
          {mouse-down in an inactive window; use SelectWindow }
          { to make it active here}
```

```
            ELSE          {mouse-down in the active window}
                DoContentClick(thisWindow, event);
        {handle other cases here}
    END; {of CASE statement}
END;        {DoMouseDown}
```

在程序清单 5-9 中，当 FindWindow 报告，有一个鼠标按下事件发生在包含控制部件的窗口的内容区域中时，DoMouseDown 调用另一个应用程序定义的过程 DoContentClick，并向它传送 FindWindow 返回的窗口指针以及事件记录。

程序清单 5-10 给出了应用程序定义的过程 DoContentClick，它使用上述信息确定鼠标按下事件是否发生在控制部件上。

程序清单 5-10 检测弹出式菜单和按钮上的鼠标按下事件

```
PROCEDURE DoContentClick (window: WindowPtr; event: EventRecord);
VAR
    mouse:                  Point;
    control:                ControlHandle;
    part:                   Integer;
    windowType:             Integer;
BEGIN
    windowType := MyGetWindowType(window);   {get window type}

    CASE windowType OF

    kPlaySoundsModelessDialogBox:
        BEGIN
            SetPort(window);
            mouse := event.where;  {get the mouse location}
            GlobalToLocal(mouse);{convert to local coordinates}
            part := FindControl(mouse, window, control);
            IF control = gSpeedPopUpControlHandle THEN
                {mouse-down in Modem Speed pop-up menu}
                DoPopUpMenu(mouse, control);
            CASE part OF
                inButton:    {mouse-down in Play button}
                    DoPlayButton(mouse, control);
                inCheckBox:{mouse-down in checkbox}
                    DoDrumRollCheckBox(mouse, control);
                OTHERWISE
                ;
            END;      {of CASE for control part codes}
```

```
        END;      {of kPlaySoundsModelessDialogBox case}
     {handle other window types, such as document windows, here}
     END; {of CASE for window types}
END; {of DoContentClick}
```

图 5-15 显示了一个“播放声音”窗口 DoContentClick 使用 FindControl 函数确定鼠标按下事件发生的地点：弹出式菜单，“播放”按钮，还是“添加单击示警声”复选框。

但是，DoContentClick 首先使用事件记录确定光标的位置，这个位置是以全局坐标指定的。因为 FindControl 函数需要窗口局部坐标的光标位置，DoContent-Click 使用 QuickDraw 的过程 GlobalToLocal 把储存在事件记录的 where 域中的点转换成当前窗口的局部坐标。GlobalToLocal 过程取一个参数，它是全局坐标的一个点，整个位图像的左上角在全局坐标系中的坐标是 (0, 0)。有关 GlobalToLocal 过程更详细的信息，请参阅《Inside Macintosh 绘图》*(Inside Macintosh: Imaging)* 一书。

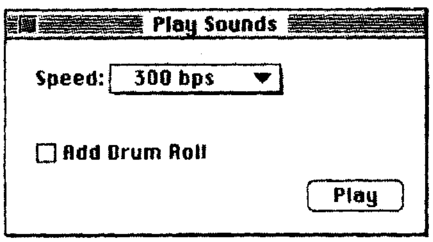


图 5-15 一个窗口中的 3 个控制部件

当 DoContentClick 调用 FindControl 时，它向 FindControl 传送以窗口局部坐标表示的光标位置，以及原来由 FindWindow 返回的指针（请参阅 5-32 页的程序清单 5-9）。

如果光标在控制部件中，FindControl 返回该控制部件的一个间接指针，以及表示控制部分的部分代码。因为弹出式控制部件定义函数不定义控制部分，DoContentClick 检测 FindControl 返回的控制间接指针，并和应用程序在它自己的全局变量中储存的弹出式菜单控制间接指针比较。如果这些是同一个控制部件的间接指针，DoContentClick 就调用应用程序定义的另一个例程 DoPopUpMenu。

在检查了 FindControl 是否返回一个弹出式菜单的间接指针后，DoContentCl-ick 使用 FindControl 返回的部分代码，确定光标是否在另两个控制部件中的一个上。如果 FindControl 返回 inButton 常数，DoContentClick 调用一个应用程序定义的例程 DoPlayButton。如果 FindControl 返回的是 inCheckBox 常数，DoContentClick 就调用应用程序定义的另一个例程 DoDrumRollCheckBox。

下一节中将要说明，所有这 3 个应用程序定义的例程，DoPopUpMenu、DoPlayButton 和 DoDrumRollCheckBox 都继续使用 TrackControl 函数跟踪并响应 FindControl 所报告的用户的鼠标在控制部件中的移动。

跟踪控制部件中的光标

在使用 FindControl 函数确定了用户按下鼠标按钮的同时，光标处在控制部件中后，应首先使用 TrackControl 函数跟踪并响应用户鼠标的移动，然后确定当用户释放鼠标按钮时，光标在控制部件的哪一部分。

您在使用 FindControl 函数确定鼠标按下事件在一个控制部件中发生后，通常要调用 TrackControl。您向 TrackControl 传送 FindControl 函数返回的控制间接指针，还向它传送和您传送给 FindControl 同样的点（即，以窗口局部坐标表示的点）。

TrackControl 函数跟踪光标在控制部件中的移动，并提供可见的反馈，直至用户释放鼠标按钮。可见的反馈取决于发生鼠标按下事件的控制部分。如果醒目显示控制部件是恰当的——例如在一个按钮中——TrackControl 就醒目显示控制部件的这个部分（并在用户释放鼠标按钮后，取消醒目显示）。如果用户按下鼠标按钮的同时，光标在一个控制部件上（例如滚动杆的滚动块），然后用户又移动鼠标，TrackControl 的响应是拖拽一个控制部件的虚线轮廓线。5-12 页的图 5-8 说明了 TrackControl 如何提供可见的反馈。

只要用户按住鼠标按钮，您也可以通过一个动作过程执行附加的动作。例如，如果用户在处理一个文字文稿，并当光标在滚动杆上时按下鼠标按钮，您的动作过程就应该连续地逐行（或以等价的度量单位）滚动文稿，直至用户释放鼠标按钮，或者到达了文稿的底部。您把这个过程的一个间接指针传送给 TrackControl。（5-57 页的“响应滚动箭头和灰度面积上的事件滚动窗口内容”将说明如何完成这个过程。）

如果用户释放鼠标按钮时，光标在控制部件中，TrackControl 函数返回控制部件的部分代码，如果用户释放鼠标按钮时，光标在控制部件之外，就返回 0。除非 TrackControl 返回 0 作为它的函数结果，您的应用程序应该适当地响应在那个控制部分中的鼠标按下事件。当 TrackControl 返回 0 作为它的函数结果时，您的应用程序不需要作任何事情。

程序清单 5-11 给出了一个应用程序定义的过程 DoPlayButton，它使用 TrackControl 跟踪图 5-15 中显示的“播放”按钮中的鼠标按下事件。DoPlayButton 例程向 TrackControl 传送 FindControl 返回的控制部件间接指针。DoPlayButton 例程还向 TrackControl 传送和它传送给 FindControl 同样的光标位置（即，一个以局部坐标表示的点）。因为按钮不需要一个动作过程，传送给 TrackControl 的最后一个参数是 NIL。

程序清单 5-11 对按钮使用 TrackControl 函数

```
PROCEDURE DoPlayButton (mouse: Point; control: ControlHandle);
BEGIN
    IF TrackControl(control, mouse, NIL) <> 0 THEN    {user clicks Play}
    BEGIN
    IF gPlayDrumRoll = TRUE THEN {user clicked Play Drum Roll checkbox }
            DoPlayDrumRoll;      { so play a drum roll first}
    SysBeep(30);{always play system alert sound when user clicks Play}
    END;
END;
```

如果用户按下鼠标按钮的同时，光标在“播放”按钮中，TrackControl 将“播放”按钮反色显示。如果用户在光标移出控制部件后释放鼠标按钮，TrackControl 停止将按钮反色显示，并返回 0，这时，DoPlayButton 不作任何事情。

但是，如果用户释放鼠标按钮时，光标在“播放”按钮中，TrackControl 停止将“播放”按钮反色显示，并返回常数 inButton 的值。然后，DoPlayButton 调用声音管理器的过程 SysBeep 播放系统示警声。(这在本书的“对话管理器”一章中说明。) 在释放鼠标按钮前，用户可以把光标移出控制部分然后又移回来，如果用户这时释放鼠标按钮，TrackControl 将仍然返回部分代码。

对于按钮、复选框、选项按钮和滚动杆中的滚动块，您的应用程序通常向 TrackControl 传送 NIL，不使用动作过程。但是，TrackControl 仍然可见地响应有效控制部件中的鼠标事件。即，如果用户按下鼠标按钮时，光标正处在一个动作过程被设定为 NIL 的控制部件，TrackControl 将适当地改变光标的显示，直至用户释放鼠标按钮。

对于滚动箭头和滚动块的灰度面积，您需要定义自己的动作过程。您把一个动作过程的指针作为一个参数传送给 TrackControl，5-57 页的“响应滚动箭头和灰度面积上的事件滚动窗口内容”一节中将有详细的说明。

对于弹出式菜单，您必须将它的动作过程的 Pointer (-1) 传送给 TrackControl；这使得TrackControl能够使用弹出式控制部件定义函数中定义的该动作过程。

当 FindControl 报告弹出式菜单中发生一个鼠标按下事件时，5-33 页的程序清单 5-10 调用一个应用程序定义的例程 DoPopUpMenu。程序清单 5-12 说明了 DoPopUpMenu 如何使用 TrackControl 处理用户在弹出式菜单中的对话。通过向 TrackControl 传送 Pointer (-1)，DoPopUpMenu 使用在弹出式控制部件定义函数中定义的动作过程。

程序清单 5-12 对弹出式菜单使用 TrackControl

```
PROCEDURE DoPopUpMenu (mouse: Point; control: ControlHandle);
VAR
    menuItem:           Integer;
    part:               Integer;
BEGIN
    part := TrackControl(control, mouse, Pointer(-1));
    menuItem := GetControlValue(control);
    IF menuItem <> gCurrentItem THEN
    BEGIN
    gCurrentItem := menuItem;
    SetMyCommunicationSpeed;{use speed stored in gCurrentItem}
    END;
END; {of DoPopUpMenu}
```

弹出式菜单的动作过程醒目显示弹出式菜单标题，显示弹出式菜单，并当用户在菜单中上下拖拽时处理和用户所有的对话。当用户释放鼠标按钮时，动作过程关闭弹出式方框，绘制用户在弹出式菜单中的选择（如果用户没有作出新的选择，就还原以前的项目），并取消弹出式标题的醒目显示。然后弹出式控制部件定义函数把控制记录的 contrlValue 域的值，更改为用户选择的菜单命令项目的编号。

因为按钮不保持设定值，所以对它们的响应也是很简单的：当用户单击按钮时，您的应用程序应该立即执行按钮标题说明的动作。对于弹出式菜单和其它类型的控制功能，您必须在响应用户的动作前，确定它们的当前设定值。例如，在响应前，您需要知道用户在弹出式菜单中选择了哪一个项目，复选框是否已标上了检查标记，或者用户把滚动块移动了多远。您采取的动作可能造成其它控制部件设定值随之改变。下一节将说明确定和更改控制部件的设定值。

在了解了如何确定和更改设定值后，请参阅 5-43 页的“滚动文稿”一节中关于如何响应滚动杆中的鼠标事件的详细讨论。

确定和更改控制部件设定状态

您的应用程序使用控制资源或者 NewControl 函数的参数，在创建控制部件时，指定了它的各种系统设定值—例如，它的当前设定值和最大及最小设定值。

但是，当用户单击一个控制部件时，您的应用程序常常需要确定该控制部件的当前设定值和其它可能的值。例如，当用户单击一个复选框时，在决定是否要在复选框内清除或绘制一个检查标记前，您的应用程序必须确定该方框是否选择过。当用户移动滚动块时，您的应用程序需要确定应该显示文稿的哪一部分。

除了响应在控制部件中的鼠标事件本身之外，应用程序必须调整一些控制部件以便响应其它的事件。例如，当用户调整窗口大小时，您的应用程序必须使用控制管理器的过程 MoveControl 和 SizeControl 适当地移动并调整滚动杆的大小。

您的应用程序可以使用 GetControlValue 函数确定控制部件的当前设定值，它还可以使用 GetControlMaximum 函数确定一个控制部件的最大设定值。

您可以使用 SetControlValue 过程更改控制部件的设定值，并相应地重新绘制控制功能。您可以使用 SetControlMaximum 过程更改控制部件的最大设定值，并重新绘制反映新设定值的指示符或滚动块。

为了响应涉及控制部件的用户动作，您的应用程序常常需要更改设定值，并可能重新绘制这个控制部件。例如，当用户点按一个复选框时，您的应用程序必须确定该复选框当前是否选定，然后交换它的设定值。当您使用 SetControlValue 交换复选框的设定值时，控制管理器在复选框内适当地选择绘制或是取消检查标记。当用户单击一个选项按钮时，您的应用程序必须确定这个选项按钮是否已经启用，如果没有，就停用原来的选项按钮，启用新选定的选项按钮。

5-34 页的图 5-15 显示了“播放声音”窗口中的一个复选框。当用户单击复选框启用它后，再单击“播放”按钮时，应用程序对它播放的声音添加单击示警声。

程序清单 5-13 给出了应用程序定义的例程 DoDrumRollCheckBox，它对单击复选框作出响应。这个例程使用 GetControlValue 函数确定复选框的上一个值，然后使用 SetControlValue 过程更改它。GetControlValue 函数返回控制部件的一个当前设定值，它被储存在控制记录的 contrlValue 域中。SetControlValue 过程把 contrlValue 设定为指定的值，并重新绘制控制部件，以便反映新的设定值。(对于复选框和选项按钮，值 1 表示向控制部件内填适当的标记，值 0 表示删除设定标记。对于滚动杆，SetControlValue 在沿着滚动杆的适当位置重新绘制滚动块。对于弹出式菜单，SetControlValue 在它的弹出式方框中显示相应于指定值的菜单命令项目的名称。)

程序清单 5-13 对点按复选框的响应

```
PROCEDURE DoDrumRollCheckBox (mouse: Point; control: ControlHandle);
VAR
     checkbox:Integer;
BEGIN
     IF TrackControl(control, mouse, NIL) <> 0 THEN{user clicks checkbox}
     BEGIN
          checkbox := GetControlValue(control);{get last value of checkbox}
          checkbox := 1 - checkbox;            {toggle value of checkbox}
```

```
        SetControlValue(control, checkbox); {set checkbox to new value}
        IF checkbox = 1 THEN              {the checkbox is checked}
        gPlayDrumRoll := TRUE  {play a drum roll next time user clicks Play}
        ELSE
            gPlayDrumRoll := FALSE;
    END;
END;
```

DoDrumRollCheckBox例程使用TrackControl确定用户选定了哪一个控制部件。当 TrackControl 报告发生用户单击复选框事件时，DoDrumRollCheckBox 使用 GetControlValue确定用户上次是选择了复选框(即控制部件的当前设定值为 1)，还是取消了复选框的选择(即控制部件的当前设定值为 0)。通过把控制功能的当前设定值减去 1，DoDrumRollCheckBox 切换到一个新的设定值，然后使用 SetControlValue 把这个新的设定值指派给复选框。SetControlValue 过程更改复选框的当前设定值，并适当地重新绘制它：如果新的设定值是 1，在方框中绘制一个 X，如果新的设定值是 0，在方框中删除 X。

5-23 页的程序清单 5-4 给出了指定窗口滚动杆的控制资源，5-24 页的程序清单 5-5 给出了应用程序的 DoNew 例程，它使用这些滚动杆创建一个文稿窗口。这个例程使用 GetNewControl 函数创建滚动杆，然后调用一个应用程序定义的例程 MyAdjust-Scrollbars。程序清单 5-14 给出了 MyAdjustScrollbars，这个例程又调用的应用程序定义的例程，确定了适当的滚动杆大小、位置和最大设定值。

程序清单 5-14 调整滚动杆的设定状态和位置

```
PROCEDURE MyAdjustScrollBars    (window: WindowPtr;
                              resizeScrollBars: Boolean);
VAR
    myData: MyDocRecHnd;
BEGIN
    myData := MyDocRecHnd(GetWRefCon(window));
    HLock(Handle(myData));
    WITH myData^^ DO
    BEGIN
        HideControl(vScrollBar);{hide the vertical scroll bar}
        HideControl(hScrollBar);{hide the horizontal scroll bar}
        IF resizeScrollBars THEN{move and size if needed}
            MyAdjustScrollSizes(window);
        MyAdjustScrollValues(window, NOT resizeScrollBars);
        ShowControl(vScrollBar);{show the vertical scroll bar}
        ShowControl(hScrollBar);{show the horizontal scroll bar}
    END;
    HUnLock(Handle(myData));
END; {of MyAdjustScrollbars}
```

当调用 DoOpen 例程在一个窗口中打开现有文稿时，SurfWriter 也使用 MyAdjustScrollbars 过程更改滚动杆的大小并进行调整。当用户更改窗口大小时，应用程序 SurfWriter 再次使用 MyAdjustScrollbars。

MyAdjustScrollbars例程首先获得窗口文稿记录的一个间接指针，文稿记录中储存了滚动杆的间接指针，以及其它有关文稿的数据。（请参阅本书的："窗口管理器"一章中有关为窗口创建应用程序自己的文稿记录的信息。）

在对滚动杆进行任何调整之前，MyAdjustScrollbars把这些控制部件的间接指针传送给控制管理器的过程HideControl，这个过程使控制部件变成不可见状态。然后，MyAdjustScrollbars 例程调用一个应用程序定义的例程 MyAdjustScrollSizes （在 5-67 页的程序清单 5-24 中给出），适当地移动滚动杆并调整它的大小。在调用另一个新的应用程序定义的例程 MyAdjustScrollValues 为滚动杆设定适当的当前和最大设定值后，MyAdjustScrollbars使用控制管理器的过程ShowControl 在新位置显示该控制部件。

程序清单 5-15 给出了 MyAdjustScrollValues 如何调用另一个应用程序定义的例程 MyAdjustHV，这个例程使用控制管理器例程为滚动杆指派适当的设定值。

程序清单 5-15 为滚动杆指派设定值

```
PROCEDURE MyAdjustScrollValues (window: WindowPtr);
VAR
    myData: MyDocRecHnd;
BEGIN
    myData := MyDocRecHnd(GetWRefCon(window));
    HLock(Handle(myData));
    WITH myData^^ DO
    BEGIN
        MyAdjustHV(TRUE, vScrollBar, editRec);
        MyAdjustHV(FALSE, hScrollBar, editRec);
    END;
    HUnLock(Handle(myData));
END; {of MyAdjustScrollValues}
```

为了防止用户的滚动过程超出文稿的边缘后看到一个空白的窗口，您应该限制滚动杆的最大设定值（请参阅 5-10 页的图 5-6）。如果窗口比文稿大（当小的文稿在大的显示器上时，很容易出现这种情况），您的应用程序应该令滚动杆最大设定值和最小设定值相等。这时，控制管理器使滚动杆无效，当所有的信息都显示在窗口内时，这样做是自然的。

程序清单 5-16 给出了应用程序定义的过程 MyAdjustHV，它是用来调整滚动杆的当前和最大设定值的。在向 isVert 参数传送了 TRUE 值后，MyAdjustHV 对垂直滚动杆计算最大和当前设定值，并进行调整；当传送的是 FALSE 时，它计算并调整水平滚动杆的当前和最大设定值。

在这个实例中，文稿由储存在 TextEdit 的编辑记录中的单一字体文字组成。TextEdit 编辑记录的 viewRect 域指定了可见文字所在的矩形；由于 viewRect 已经排除了滚动杆区域，MyAdjustHV 在计算这些滚动杆的最大设定值时，不需要从窗口高度或宽度中减去滚动杆区域。[有关 TextEdit 更详细的信息，特别是编辑记录的信息，请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书。]

程序清单 5-16 调整滚动杆的最大和当前设定值

```
PROCEDURE MyAdjustHV (isVert: Boolean; control: ControlHandle;
                       editRec: TEHandle);
VAR
    oldValue, oldMax, width:   Integer;
    max, lines, value:         Integer;
BEGIN
    {calculate new maximum and current settings for the vertical or }
    { horizontal scroll bar}
    oldMax := GetControlMaximum(control);
    oldValue := GetControlValue(control);
    MyGetDocWidth(width);
    IF isVert THEN    {adjust max setting for the vertical scroll bar}
    BEGIN
    lines := editRec^^.nLines;
    {since nLines isn't right if the last character is a carriage }
    { return, check for that case}
    IF Ptr(ORD(editRec^^.hText^) + editRec^^.teLength - 1)^ = kCRChar THEN
            lines := lines + 1;
    max := lines - ((editRec^^.viewRect.bottom - editRec^^.viewRect.top)
            DIV editRec^^.lineHeight);
    END
    ELSE          {adjust max setting for the horizontal scroll bar}
        max := width - (editRec^^.viewRect.right - editRec^^.viewRect.left);
    IF max < 0 THEN
        max := 0;         {check for negative settings}
    SetControlMaximum(control, max);     {set the max value of the control}
    IF isVert THEN    {adjust current setting for vertical scroll bar}
        value := (editRec^^.viewRect.top - editRec^^.destRect.top)
                DIV editRec^^.lineHeight
```

```
    ELSE          {adjust current setting for the horizontal scroll bar}
        value := editRec^^.viewRect.left - editRec^^.destRect.left;
    IF value < 0 THEN
        value := 0
    ELSE IF value > max THEN
    value := max; {don't allow current setting to be greater than the }
                  { maximum setting}
    SetControlValue(control, value);
END; {of MyAdjustHV}
```

MyAdjustHV 例程首先使用 GetControlMaximum 和 GetControlValue 函数确定要调整的滚动杆的最大和当前设定值。

然后，MyAdjustHV 对垂直滚动杆计算一个新的最大设定值。因为窗口显示的是一个只有文字的文稿，MyAdjustHV 使用编辑记录的 nLine 域确定文稿中的总行数，因而也就确定了文稿的长度。然后，MyAdjustHV 从文稿长度中减去窗口的高度，并令这个值为垂直滚动杆的最大设定值。

为了以像素为单位计算窗口的总高度，MyAdjustHV 先从显示矩形的底部坐标减去它的顶部坐标。(正常情况下，窗口左上角的坐标是 [0，0]；因此，矩形底部一点的垂直坐标的值比矩形顶部一点的垂直坐标大。) 然后，MyAdjustHV 用窗口高度 (以像素为单位) 除以编辑记录的 lineHeight 域的值，对于单一字体文字，该域的值指定了文稿行的高度 (以像素为单位)。MyAdjustHV 通过这个方法，确定了窗口的高度 (以行数为单位)。

MyAdjustHV 例程使用另一个应用程序定义的例程 MyGetDocWidth 确定文稿的宽度。为了计算窗口的宽度，MyAdjustHV 从显示矩形的右坐标减去它的左坐标。MyAdjustHV 从文稿宽度中减去窗口宽度，就得到了水平滚动杆的最大设定值。

只要窗口比文稿大，MyAdjustHV 对垂直和水平滚动杆的最大设定值都指定为 0。例如，当为一个尚不包含数据的新文稿创建窗口时，MyAdjustHV 为滚动杆设定的最大和当前设定值都是 0。当最大和最小设定值相等时，滚动杆的标准控制部件定义函数自动地使滚动杆无效。这当然是完全正确的，因为当用户不需要滚动的内容时，滚动杆应该是无效的。当您使最大设定值超过最小设定值时，控制部件定义函数就使滚动杆重新有效。

然后，MyAdjustHV 例程使用控制管理器的过程 SetControlMaximum 把新计算的最大设定值指派给每个滚动杆。SetControlMaximum 过程修正控制部件，以便反映新的最大设定值；例如，如果用户删除了文稿的一大部分，最大设定值就要减小，SetControlMaximum 移动滚动块，指示相对于较小文稿的新位置。

当用户在文稿中添加或删除信息，或者调整窗口大小时，您的应用程序也可能需要调整滚动杆的当前设定值，MyAdjustHV 例程对控制部件计算一个新的当前设定值，然后使用 SetControlValue 把该设定值指派给控制部件，并相应地将滚动块重新定位。

在编辑记录的destRect 域中指定的目的矩形是文字在其中绘制的矩形，而查看矩形是文字在其中确实可见的矩形。MyAdjustHV 从目的矩形的顶部坐标减去查看矩形的顶部坐标，并把结果除以行高，就得到了当前在窗口顶部显示的行数。MyAdjustHV 把这个行数用作垂直滚动杆的当前设定值。

为了导出水平滚动杆的当前设定值(以像素为单位)，MyAdjustHV从显示矩形的左坐标减去目的矩形的左坐标。

滚动文稿

本章的前面各节说明了如何创建滚动杆、如何确定鼠标按下事件在滚动杆发生的时间、跟踪用户在滚动杆中的动作，以及确定并更改滚动杆的设定值。本节讨论您的应用程序如何响应用户在滚动杆中的鼠标活动滚动文稿。例如，在下列条件下，您的应用程序向文稿的底部滚动文稿:

- 当用户把滚动块拖拽到垂直滚动杆的底部时，您的应用程序应该显示用户文稿的结尾
- 当用户单击滚动块下面的灰度面积时，您的应用程序应该把文稿向上移，显示指向文稿底部的下一个窗口的信息，同时调用 SetControlValue 移动滚动块
- 当用户单击向下滚动箭头时，您的应用程序应该把文稿向上移一行（或某些类似的度量单位)，并把文稿下部的内容移人显示矩形，它应该使用 SetControlValue 移动滚动块

您的应用程序的第一步，必须是决定滚动的距离。当用户把滚动块拖拽到滚动杆上的新位置时，您的文稿也滚动到一个相应的位置。

当用户单击一个滚动箭头时，您的应用程序确定应该滚动的量。一般来说，一个文字处理程序在垂直方向滚动一行文字，水平方向滚动一个平均字符宽度，数据库或电子表格滚动一个域。图形应用程序应该尽可能地滚动显示整个对象。(通常，在使用控制管理器、QuickDraw 和 TextEdit 的例程时，应用程序把这些距离转换成像素。)

当用户单击滚动杆的灰度面积时,您的应用程序滚动的距离应该恰好比窗口的高度和宽度小。为了确定这个高度和宽度，您可以使用滚动杆控制记录的contrlOwner域。这个域包含窗口记录的一个指针。当您滚动一个窗口的距离，最好保留上一个窗口的一小部分。这个保留的部分能够帮助用户跟踪上下文。例如，如果用户在文字文稿中向下滚动一个窗口的距离，窗口顶部的那行文字应该是上一个窗口底部出现的那一行。

滚动方向是根据滚动距离是正数还是负数决定的。当用户向下或向右滚动时，卷动距离是负数，向上或向左滚动的距离是正数。例如，当用户从文稿的开始滚动到位于向下 200 个像素的一行时，垂直滚动杆上的滚动距离为 -200 个像素。当用户从那里滚动回文稿开始处时，滚动距离是 200 个像素。

确定滚动距离只是第一步。简而言之，为了响应用户对滚动杆的操纵，您的应用程序应该采取下列步骤滚动文稿:

1. 应使用 FindControl、GetControlValue 和 TrackControl 函数帮助计算滚动距离。

2. 如果您在进行的滚动不是由于用户拖拽滚动块，就应该使用 SetControlValue 过程把滚动块移动一个相应的量。

3. 应使用一个例程—例如 QuickDraw 的过程 ScrollRect 或 TextEdit 的过程 TEPinScroll—把窗口中显示的部分移动一个距离，它等于计算出的滚动距离。然后使用生成更新事件的调用或直接调用应用程序的 DoUpdate 例程，执行其余步骤。

4. 应使用 UpdateControls 过程更新滚动杆，然后调用窗口管理器的过程 DrawGrowIcon 重新绘制尺寸格。

5. 应使用 QuickDraw 的过程 SetOrigin 把窗口的原点更改一个量，这个量等于滚动杆的设定值，这样文稿的左上角就在窗口局部坐标的 (0, 0) 坐标。(执行了这一步，可以使您的应用程序的文稿绘制例程在窗口的正确面积进行绘制。)

6. 应调用您的应用程序的例程在窗口内重新绘制文稿。

7. 应使用 SetOrigin 过程把窗口的原点复位到 (0, 0)，使得以后的窗口管理器和控制管理器的例程都能够在窗口的正确面积进行绘制。

8. 返回您的事件循环。

本节的其余部分将对这些步骤进行详细的解释。

附注

本章后面介绍的SetOrigin不是必须要使用的。当您更新窗口时，此过程仅可以通过滚动杆的当前设定状态帮助您偏移窗口的起始位置，因此，您可以使用一种坐标系统（其文稿的左上角总是处于（0，0））来定位文稿中的对象。作为此方法的替代方法，您的应用程序可以保持窗口的左上角（称为窗口起始位置）位于（0，0），而不是用等于滚动杆的设定状态的一个总量偏移您文稿中的项目。如果您倾向于这种替代方法，那么就可以使用在《inside Macintosh 绘图》*(inside Macintosh : Imaging)* 中介绍的 QucikDraw 过程 OffsetRect、OffsectRgnSubpt 和 Addpt。

当用户存储一个文稿时，您的应用程序应把数据存储于您自己的应用程序定义的数据结构中。（例如，本章中的示例代码将 TextEdit 编辑记录的间接指针存储在 document record 中。编辑记录中包含有关文字的信息，如它的长度和它自己的局部坐标系统、以及用于文字本身的间接指针。）您一般可使用文稿的局部坐标（文稿的左上角位于（0，0））来存储您的应用程序显示在屏幕上的对象的信息。

图5 - 16的左侧表示用户已打开一个现有的文稿，示例应用程序 SurfWriter 显示出该文稿的顶部。在这个实例中，文稿中包含35行单一字形的文字，全文的行高度是10像素。因此，文稿的长度为350像素。当用户首次打开该文稿时，窗口的起始位置与文稿空间的左上点一致：都位于（0，0）。

在这个实例中，窗口显示了15行文字，总计150像素。因为垂直滚动杆的最大设定状态是文稿长度减去其窗口的高度，所以，该滚动杆的最大设定状态是 200 像素。

假设用户将滚动块向下拖拽到垂直滚动杆的中间。由于用户希望向下滚动，SurfWriter 应用程序为了显示文稿底部的更多内容，就必须将文稿的文字向上移动。为了响应用户请求的向下滚动，就需要利用一个带有负数值的滚动距离将文稿向上移动。（同样，为了响应用户请求的向上滚动，就需要利用一个带有正数值的滚动距离将文稿向下移动。）

应用程序 SurfWriter 可以利用 FindControl、TrackControl 和 GetControlValue 来确定它必须将文稿向上移动100像素，即移动 - 100像素的滚动距离。（5 - 53 页的“为响应滚动块中的事件而滚动屏幕”中详细解释了如何使用 FindControl、TrackControl 和 GetControlValue 来确定滚动距离。）

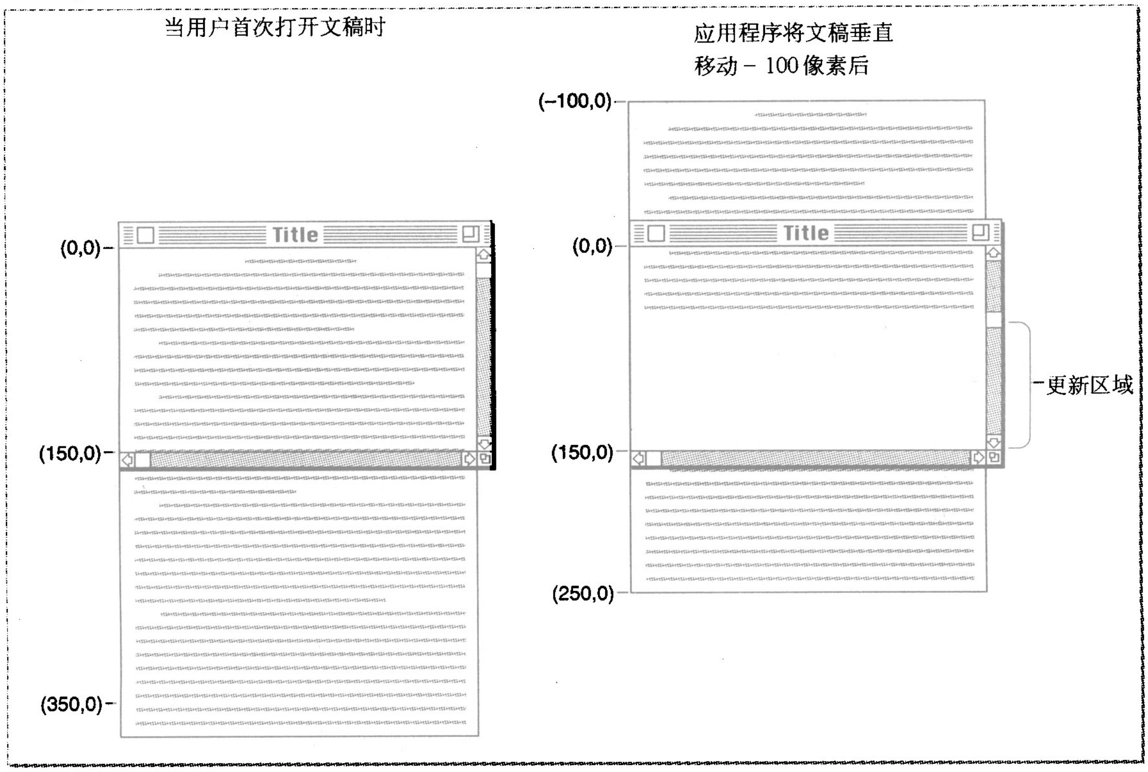


图 5 - 16　相对窗口移动文稿

然后，应用程序 SurfWriter 利用 QuickDraw 的过程 ScrollRect 以 - 100 像素的距离移动窗口中显示的各部分。ScrollRect 过程将文稿向上移动 100 像素即 10 行；原窗口显示的最下面 5 行现在出现在窗口的顶部，并且应用程序 SurfWriter 将该窗口的其余部分添加到一个更新区域中，用于以后的更新操作。

ScrollRect 过程不更改窗口的坐标系统；而是将窗口中的各个位移动到新的坐标中，这些坐标仍在窗口的局部坐标系统中。为了达到更新窗口的目的，您可以把这种作法看作是改变整个文稿的坐标，如图 5 - 16 右侧所示。

ScrollRect 过程需要 4 个参数：需滚动的矩形、滚动的水平距离、滚动的垂直距离和区域间接指针。通常，当指定了要滚动的矩形时，您的应用程序就传送一个值，该值表示内容区域减去滚动杆区域所得的值，如程序清单 5 - 17 所示。

程序清单 5 － 17 使用 ScrollRect 滚动窗口中显示的各部分

```
PROCEDURE DoGraphicsScroll (window: WindowPtr;
                            hDistance, vDistance: Integer);
VAR
    myScrollRect: Rect;
    updateRegion: RgnHandle;
BEGIN
    {initially, use the window's portRect as the rectangle to scroll}
    myScrollRect := window^.portRect;
    {subtract vertical and horizontal scroll bars from rectangle}
    myScrollRect.right := myScrollRect.right - 15;
    myScrollRect.bottom := myScrollRect.bottom - 15;
    updateRegion := NewRgn;         {always initialize the update region}
    ScrollRect(myScrollRect, hDistance, vDistance, updateRegion);
    InvalRgn(updateRegion);
    DisposeRgn(updateRegion);
END; {of DoGraphicsScroll}
```

重要告诫

首先您必须将水平距离作为参数传送到 ScollRect 中，然后再传送垂直距离。请注意，当您指定 QuickDraw 坐标系统中的一个点时，次序恰恰相反：您应该首先命名垂直坐标，其次命名水平坐标。

虽然尺寸较短一侧的滚动杆都是 16 像素，但在程序清单 5 － 17 所示的 DoGraphicsScroll 过程中只减去15像素，因为滚动杆的边缘与窗口框架的边缘重叠，使滚动杆在窗口的内容区域中只占据 15 个像素。

由 ScrollRect 移出矩形（该矩形由 myScrollRect 指定）之外的那一部分不在屏幕上绘制出来，并且也不存储它们－－您的应用程序负责跟踪这些数据。

ScrollRect 过程将这些位水平移动的距离是 hDistance 像素，垂直移动的距离 vDistance 像素；当 DoGraphicsScroll 在这些参数中传送正数时，ScrollRect 在 myScrollRect 参数中把各部分分别向右和向下移动。当用户希望向左或向上滚动时这是一种恰当的作法，因为当应用程序 SurfWriter 完成更新窗口时，用户便可分别延左边和上边看到文稿更多的内容。(请记住：若要向上或向左滚动，就应向下或向右移动文稿，二者均为正方向。)

当 DoGraphicsScroll 传送的这些参数是负数时，ScrollRect 将 myScrollRect 参数中的各个位分别向左或向上移动。当用户希望向右或向下滚动时可采取这种做法，因为当应用程序 SurfWriter 完成更新窗口时，用户便会延右边或下边看到更多的文稿内容。(请记住：若要向下或向右滚动，文稿就应向上或向左移动，二者均为反方向。)

在图5 - 16中，应用程序SurfWriter确定了垂直滚动距离是 - 100，它是在如下所示vDistance参数中传送的:

```
ScrollRect(myScrollRect, 0,-100, updateRegion);
```

但是，如果用户希望在此点上将滚动块移回到文稿的开始处，应用程序便确定向上滚动的距离是像素，并且在vDistance参数中传送正数值100。

在使用ScrollRect将已存在于窗口中的各部分移动以后，应用程序SurfWriter应该利用其标准窗口更新代码在窗口的更新区域中绘制这些内容。

根据前面的解释，ScrollRect事实上更改了相对于窗口局部坐标的文稿坐标。就窗口的局部坐标系统而言，该文稿的左上角目前位于(- 100, 0)，如图5 - 16的右侧所示。若要简化更新窗口的操作，应用程序SurfWriter可以使用QuickDraw的过程SetOrigin来更改窗口的坐标系统，这样，应用程序SurfWriter就能够把文稿的左上角仍看成是位于（0, 0)。

SetOrigin过程需要2个参数: 首先是窗口起始位置的新的水平坐标，其次是窗口起始位置的新的垂直坐标。

重要告诫

像ScrollRect一样，SetOrigin也需要您依次传送一个水平坐标和一个垂直坐标。请注意，当您在QuickDraw坐标系统中指定一个点时，次序恰恰相反: 您应该首先命名垂直坐标，其次命名水平坐标。

只要您做好更新窗口的准备（如进行滚动以后)，您就可以使用GetControlValue来确定水平滚动杆的当前设定状态，并且传送此值，将它作为窗口起始位置的新的水平坐标。然后使用GetControlValue来确定垂直滚动杆的当前设定状态，并且传送此值，将它作为窗口起始位置的新的垂直坐标。当您在文稿的窗口内重新绘制文稿时，用这种方式使用SetOrigin来移动窗口的局部坐标系统就可以使文稿的左上角总是位于（0, 0)。

例如，在用户操纵垂直滚动杆移动到距文稿顶部100象素（或向上或向下）的位置以后，应用程序SurfWriter进行如下的调用操作:

```
SetOrigin(0, 100);
```

虽然滚动距离是 - 100，(是相对距离)，但滚动杆的当前设定状态目前是100。(因为您在QuickDraw坐标系统中首先应指定点的垂直坐标，其次指定水平坐标，所以SetOrigin中参数的顺序最初可能会造成混淆。)

从图5 - 17的左侧可以看出应用程序SurfWriter是如何使用SetOrigin过程将窗口起始位置移动到点（100，0），以便使文稿左上角位于当前的窗口局部坐标系统中(0，0)的位置上。这样可以还原该文稿的起始坐标空间，并使应用程序易于在窗口的更新区域中进行绘制。

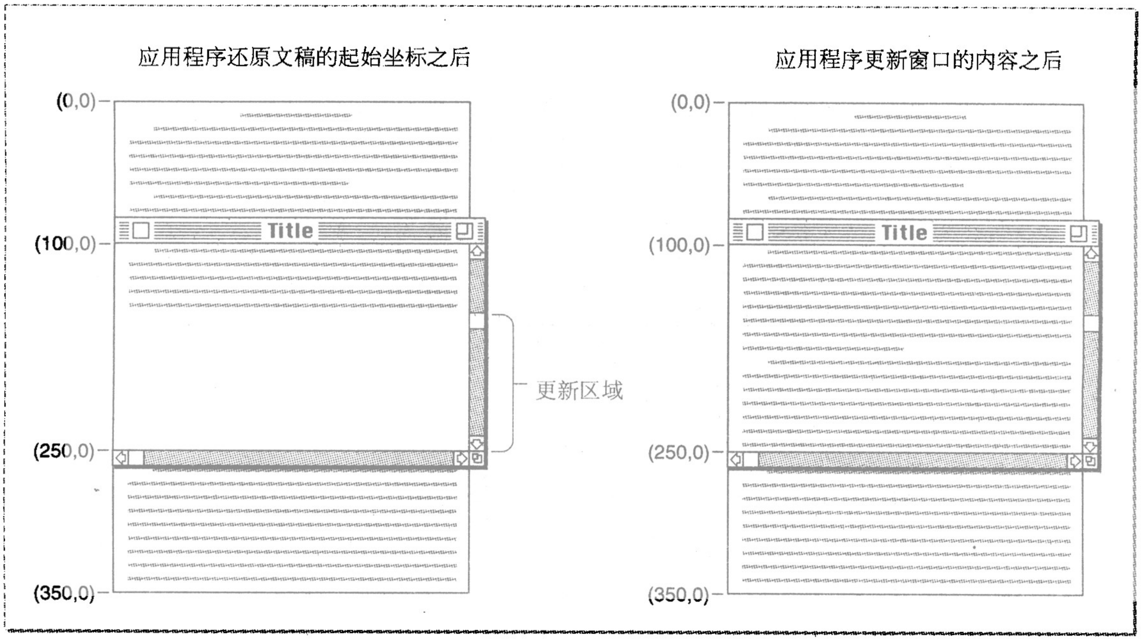


图5 - 17　更新经滚动的窗口的内容

在还原了文稿的起始坐标以后，应用程序SurfWriter便更新窗口，如图5 - 17右侧所示。应用程序所绘制的第16至第24行在文稿记录中存储的位置是从（160，0）到(250，0)。

让我们回顾一下这种作法的步骤 用户将滚动块向下拖拽了垂直滚动杆一半的距离 应用程序SurfWriter确定此距离等于 - 100像素的滚动距离；应用程序SurWriter将此距离传送到ScrollRect中，该过程可将窗口中的各部分向上移动100像素，并且在窗口的其余部分中创建一个更新区域 应用程序SurfWriter将参数中的垂直滚动杆当前设定状态（100像素）传送到SetOrigin中，以便重新绘制窗口更新区域时可以使用文稿的局部坐标；最后，应用程序SurfWriter在窗口的更新区域中绘制文字。

但是，窗口的起始位置不能置于（100，0），因此，应用程序SurfWriter在它执行自己的绘制操作以后，调用SetOrigin将窗口的起始位置复位到（0，0），因为当窗口管理器和控制管理器绘制一个窗口时，它们通常假定窗口的左上点是在（0，0）。图5 - 18说明应用程序如何按照其窗口更新例程的计算结果，使用SetOrigin将窗口起始位置设定回（0，0）。在完成更新操作以后，应用程序便再次在事件循环中处理事件。

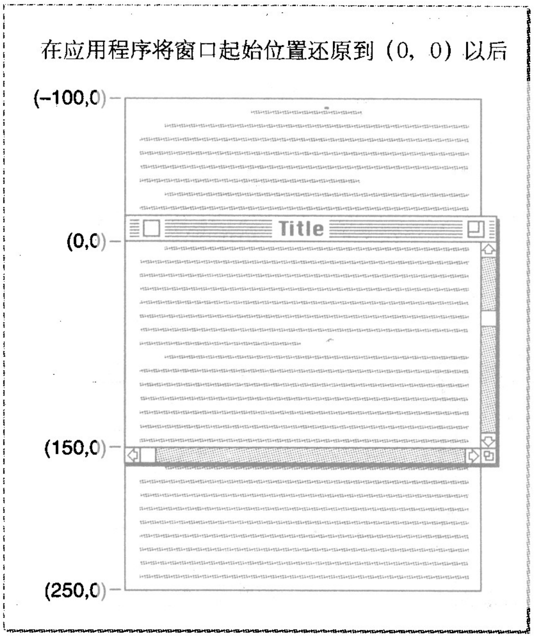


图5－18　窗口起始位置还原到（0，0）

从图5－19的左侧可以看出，当用户滚动文稿直至结尾部分时，会出现另外10行（或者100像素）的距离。在应用程序SurfWriter调用ScrollRect以后，原窗口显示中的最下面5行出现在新窗口的顶部，而该窗口下面的部分成为新的更新区域。因为用户已经滚动了总数为200像素的距离，所以应用程序使用SetOrigin将窗口起始位置更改成（200，0），如图5－19的右侧所示。

从图5－20的左侧可以看出应用程序SurWriter在窗口的更新区域中绘制的情况；从右侧可以看出应用程序SurfWriter将窗口起始位置还原到（0，0）。

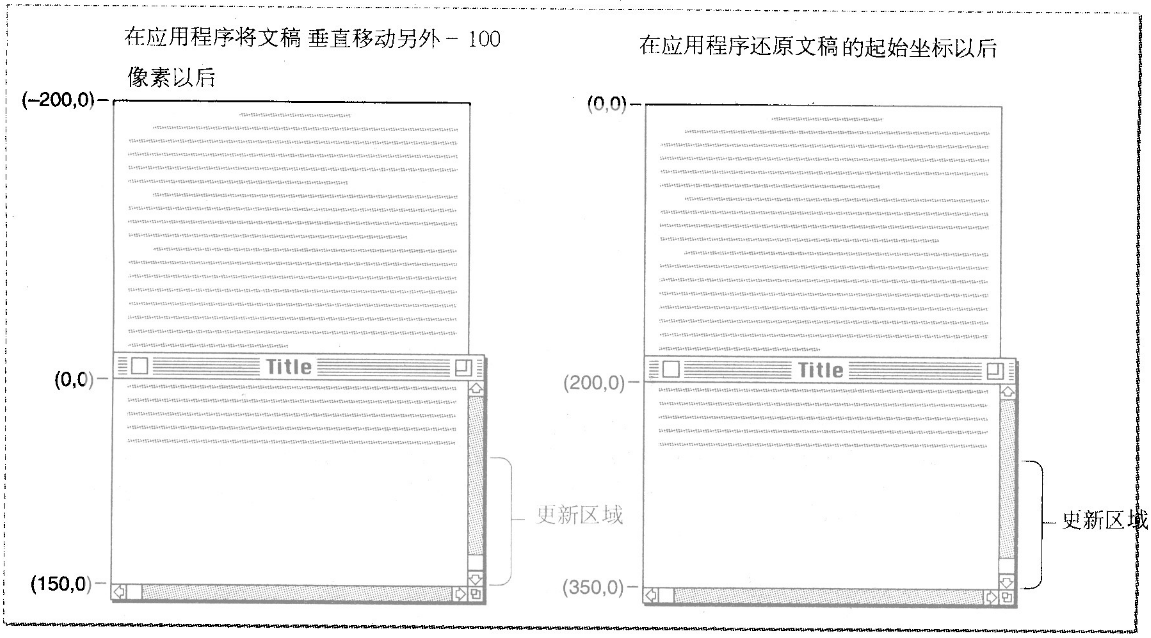


图 5 － 19　　滚动到文稿结尾部分

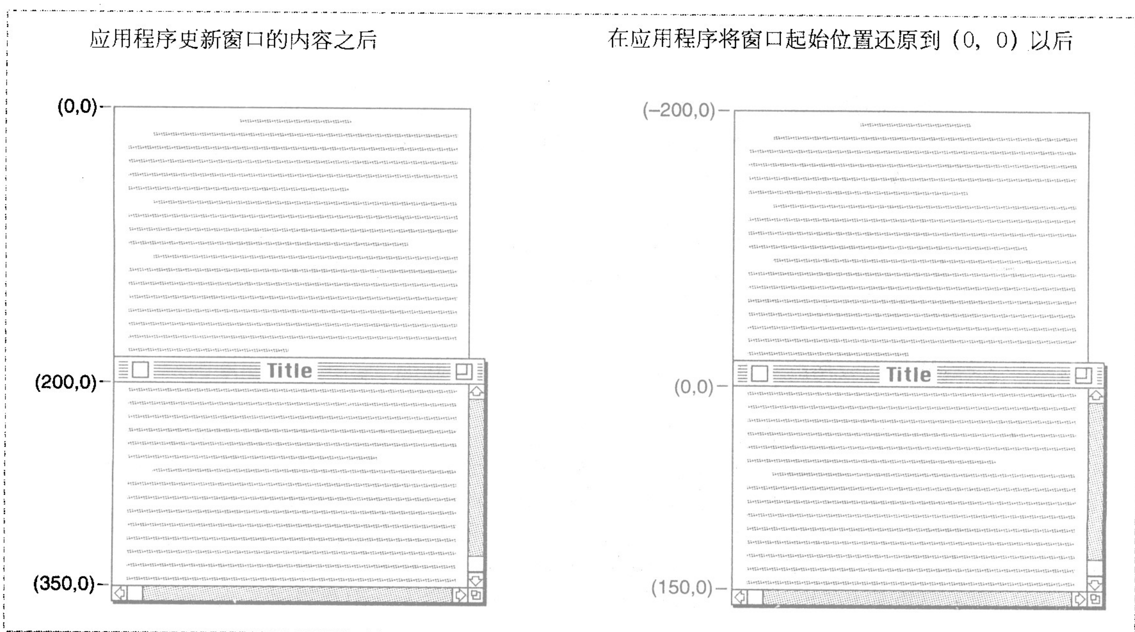


图 5 － 20　　更新窗口内容和将窗口起始位置返回到（0，0）

在下面两节“为响应滚动块中的事件而滚动屏幕”和“为响应滚动箭头和灰度面积中的事件而滚动屏幕”中，更为详尽地解释了您的应用程序如何确定滚动距离和如何以此距离移动窗口中的各部分。在这两节的后面是“在窗口内绘制滚动后的文稿”一节，介绍了您的应用程序应该如何编写其窗口更新代码段，以便在滚动后的窗口中进行绘制。关于 SetOrigin 和 ScrollRect 过程的详细说明，请参阅《Inside Macintosh 绘图》(*Inside Macintosh : Imaging*)。

至此，我们的讨论中均假设您对用户操纵滚动杆予以响应而进行滚动操作。大多数时候，由用户确定何时滚动以及滚动到何处。但是，除了用户操纵滚动杆以外，在下列四种情形下，您的应用程序也必须对文稿进行滚动。在您设计应用程序时必须考虑到这几种情况。

■ 当您的应用程序执行一个操作，而该操作可能附带产生一个新的选定项或移动插入点时，您应该滚动屏幕来显示新的选定项。例如，当用户请求一个搜索操作时，您的应用程序就要寻找所需要的文字。如果该文字位于当前的不可视的文稿部分中，您就应该滚动屏幕来显示该选定项。在用户请求贴操作以后，您的应用程序也需要进行滚动。只需插入点出现在被贴部分的尾部之后，就要进行滚动，直到看得见该选定项和新的插入点为止

■ 当用户从键盘将信息键入到窗口的边缘时，您应该滚动屏幕，将新的信息插入并显示出来。因为用户关注于新的信息，所以不应保留文稿的位置并记录用户看到的新信息。通常，字处理程序是滚动一行文字，数据库或电子表格是滚动一个域。图形应用程序应该尽可能滚动屏幕来显示整个对象。否则，您应该确定您的应用程序在滚动时重新绘制窗口内容的速度，以及为尽量减少闪烁和屏幕刷新而调整滚动操作的速度。请尽量确保滚动速度适宜，使用户既不感到过慢而妨碍操作，也不感到过快而导致混乱

■ 当用户按住鼠标按钮并将光标移过窗口的边缘，以便扩展选定项时，您应该向光标移动的方向滚动窗口。滚动的速度可以如同用户在对应的滚动箭头上按住鼠标按钮。在某些情况下，当用户将光标向远离窗口边缘的方向移动时，滚动速度可以根据光标的移动速度而变化

■ 有时用户选定某项、滚动到新的位置，然后试图执行该选定项的操作。在这种情况下，您应该滚动屏幕，以便在您的应用程序执行该操作之前显示出该选定项。显示出选定项可以使用户清楚被更改的内容

当设计您的应用程序的文稿滚动例程时，亦请参阅如下的用户界面准则：

■ 当您的应用程序自动滚动屏幕时，应该避免不必要的滚动。由于用户希望控制文稿的位置，所以您的应用程序应该仅仅根据需要来移动文稿。因此，如果选定项部分已经在窗口中显示出来，就完全不必滚动屏幕。此规则仅有一个例外，就是当选定项的隐藏部分比可视部分更重要时；此时可滚动屏幕来显示出重要的部分。例如，假设用户选定一个大的文字块，并且仅有其底部是当前可视的。如果用户这时键入一个字符，您的应用程序就必须滚动到新键入字符的位置上，用户可以看到它们

- 如果您的应用程序可以延一个方向滚动来展现选定项，就不要同时延两个方向滚动。也就是说，如果您可以垂直滚动来显示选定项，就不要再水平滚动

- 当您可以显示选定项两边的上下文时，这是一种有效的作法。此外将选定项定位于靠近窗口中部的区域比定位于窗口的角落更为适宜。当选定项太大，不能全部放入窗口中时，如果将未选定的信息放在选定项的开头或结尾处显示则有助于提供上下文

为响应滚动杆中的事件进行滚动

5－30页的“在控制部件中响应鼠标事件”中简单地介绍了如何在您的事件代码段中使用FindControl和TrackControl。从程序清单5－18中可以看到如何使用这些例程以分别响应滚动杆中的鼠标事件。

程序清单 5 － 18 响应滚动杆中的鼠标事件

```
PROCEDURE DoContentClick (window: WindowPtr; event: EventRecord);
VAR
     mouse:               Point;
     control:             ControlHandle;
     part:                Integer;
     myData:              MyDocRecHnd;
     oldSetting:          Integer;
     scrollDistance:      Integer;
     windowType:          Integer;
BEGIN
     windowType := MyGetWindowType(window);
     CASE windowType OF
          kMyDocWindow:
               BEGIN
                    myData := MyDocRecHnd(GetWRefCon(window));
                    HLock(Handle(myData));
                    mouse := event.where;
                    GlobalToLocal(mouse);    {convert to local coordinates}
                    part := FindControl(mouse, window, control);
                    CASE part OF
                    {handle all other parts first; handle scroll bar parts last}
                    inThumb:                 {mouse-down in scroll box}
```

```
        BEGIN    {get scroll bar setting}
            oldSetting := GetControlValue(control);
            {let user drag scroll box around}
            part := TrackControl(control, mouse, NIL);
            {until user releases mouse button}
            IF part = inThumb THEN
            BEGIN        {get new distance to scroll}
            scrollDistance := oldSetting - GetControlValue(control);
                IF scrollDistance <> 0 THEN
                    IF control = myData^^.vScrollBar THEN
                        TEPinScroll(0, scrollDistance *
                                myData^^.editRec^^.lineHeight,
                                    myDta^^.editRec);
                ELSE
                TEPinScroll(scrollDistance, 0, myData^^.editRec);
                    END; {of handling mouse-up in scroll box}
                END; {of handling mouse-down in scroll box}
            inUpButton, inDownButton, inPageUp, in PageDown:
            {mouse-down in scroll arrows or gray areas}
                IF control = myData^^.vScrollBar THEN
                                {handle vertical scroll}
        part := TrackControl(control, mouse, @MyVerticalActionProc)
                ELSE         {handle horizontal scroll}
        part := TrackControl(control, mouse, @MyHorzntlActionProc);
            OTHERWISE                    ;
            END; {of CASE part}
            HUnLock(Handle(myData));
        END; {of kMyDocWindowType}
        {handle other window types here}
    END; {of CASE windowType}
END;
```

当光标位于一个可视的现用滚动块上时，如果用户按下鼠标按钮，则作为其结果，FindControl 将返回用于滚动块的部分代码。以下是用来代表滚动块的部分代码和常数:

常数	**部分代码**	**控制部分**
滚动块	129	见原文

如程序清单 5 - 18 所示，当 `FindControl` 返回 `inThumb` 值时，您的应用程序应该立即调用 `GetControlValue` 来确定滚动块的当前设定状态。如果用户拖拽滚动块，您就将此设定状态减去新的当前设定状态(当用户释放鼠标按钮时，新的当前设定状态变为有效)，并把此结果作为您滚动的距离。

在使用 GetControlValue 确定了滚动块的当前设定状态以后，您可使用 TrackControl 来跟随滚动块内的光标移动，并拖拽滚动块的虚线轮廓以响应用户所做的拖动。

当用户释放鼠标按钮时，如果光标仍在滚动块中，TrackControl 便返回 inThumb 值，如果光标在滚动块外，TrackControl 便返回 0。当 TrackControl 返回 0 时，您的应用程序不予响应。否则，您的应用程序便再次调用 GetControlValue 来计算要滚动的距离。

通过调用GetControlValue和从原滚动杆设定状态减去新的滚动杆当前设定状态就可以计算出要滚动的距离，在用户释放鼠标按钮以前用GetControlValue 可以确定出原滚动杆的设定状态。如果此距离不为0，您就应该以该距离移动窗口中的各个控制部件，并且更新窗口其余部分的内容。

在滚动屏幕以前，您必须确定该滚动杆是一个垂直滚动杆还是水平滚动杆。根据本章前面部分的解释，您应该在一个文稿记录中存储滚动杆的间接指针，其中的一个间接指针是您为每个文稿创建的。通过对比包含有垂直滚动杆间接指针的域，您就可以确定由 FindControl 返回的控制部件间接指针是不是垂直滚动杆的间接指针。如果是，说明用户移动了垂直滚动杆的滚动块。如果不是，说明用户移动了水平滚动杆的滚动块。

在确定用户拖拽了哪个滚动杆的滚动块以后，您就可以将窗口中的文稿内容移动相应的滚动距离。也就是说，对垂直滚动杆中的滚动距离若为正数，就将窗口中的各部分向下移动该距离。当您更新窗口时，就会显示出文稿顶部的更多行——这就是当用户向上（UP）移动滚动块时的相应情况。水平滚动杆中的滚动距离若为正数，就将窗口中的各部分向右移动该距离。当您更新窗口时，就会显示出文稿左侧的更多行——这就是当用户向左（to the left）移动滚动块时的相应情况。（请记住：若要向上或向左滚动，文稿就向下或向右移动，两个均为正方向。）

垂直滚动杆中的滚动距离若为负数（如第 5 － 46 页图 5 － 16 所示），就将窗口中的各部分向上移动该距离。当您更新窗口时，就会显示出文稿底部的更多行——这就是当用户向下（Down）移动滚动块时的相应情况。水平滚动杆中的滚动距离若为负数，就将窗口中的各个位向左移动该距离。当您更新窗口时，就会显示出文稿右侧的更多行——这就是当用户向右移动（to the right）滚动块时相应的情况。（请记住：若要向下或向右滚动，就将文稿向上或向左移动，两个均为负方向。）

在本章先前所举的实例中，可以看到应用程序使用一个 TextEdit 编辑记录来存储由用户创建的单一字形文字。为了简化文字处理的需要，TextEdit 提供的许多例程可以简化您的工作；例如，TEPinScroll 以您的应用程序所指定的像素数量来滚动一个编辑记录的显示矩形中的文字；当最后一行滚动到显示矩形中时，TEPinScroll 便停止滚动。

TEPinScroll 过程需要 3 个参数：水平移动文字的像素数量、垂直移动文字的像素数量和一个编辑记录的间接指针。前两个参数是正数值时；分别向右和向下移动文字；是负数值时，则分别向左和向上移动文字。

在 5 – 53 页的程序清单 5 – 18 中，DoContentClick 过程将用于 TEPinScroll 的第2个参数中，并且将用于垂直滚动杆的滚动距离传送到水平滚动杆的滚动距离即传送到第 1 个参数中。

5 – 41 页上的程序清单 5 – 16 表示一个应用程序定义的例程 MyAdjustHV，只要示例应用程序 SurWriter 创建、打开、或调整窗口大小，便调用此例程。该例程以文字的行数来定义垂直滚动杆的当前和最大设定状态。

在第 5 – 53 页上的 DoContentClick 过程使用 GetControlValue 来确定控制部件的当前设定状态—对于垂直滚动杆，DoContentClick 以行数计算。当确定垂直滚动杆的滚动距离时，DoContentClick 则计算出一个代表行数的值。

但是，TEPinScroll 在其参数中仅接受传送来的像素，而不接受行数。因此 DoContentClick 将滚动距离（该过程以文字的行数计算）乘以行高度（保留在单一字形文字的编辑记录中的像素值）。这样，DoContentClick 以像素为单位将滚动距离传送到 TEPinScroll 中，如以下代码片段所示：

```
IF control = myData^^.vScrollBar THEN
TEPinScroll(0, scrollDistance * myData^^.editRec^^.lineHeight,
                    myData^^.editRec);
```

第 5 – 46 页图 5 – 16 所示的滚动距离是 – 10 行。如果行高度是 10 像素，应用程序 SurfWriter 将 – 100 作为第二个参数传送到 TEPinScroll 中。

TEPinScroll 过程将滚动离开面积添加到更新区域中，并且生成一个更新事件，使得在编辑记录的显示矩形中的文字被更新。然后，示例应用程序 SurfWriter 在其处理窗口更新事件的代码段中使用 TEUpdate 过程（请参阅从第 5 – 62 页的“在窗口内绘制滚动后的文稿”中的介绍）来处理包含有 TextEdit 记录的窗口。

若要详细了解 TEPScroll、TextEdit 编辑记录和 TextEdit 提供的其它功能，请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)*。

QuickDraw 中的过程 ScrollRect 是较通用的一个例程，当滚动屏幕时用于移动窗口中的各个位。如果您使用 ScrollRect 滚动窗口中显示的各部分，您就应该定义并使用一个 DoGraphicsScroll 例程（如第 5 – 47 页程序清单 5 – 17 所示），而不使用第 5 – 53 页程序清单 5 – 18 用到的 TEPinScroll 例程。

ScrollRect 过程在 updateRegion 参数中返回需要更新的面积。然后，DoGraphicsScroll 过程（如第 5 – 47 页程序清单 5 – 17 所示）使用 QuickDraw 的过程 InvalRgn 将此面积添加到更新区域中，强制生成一个更新事件。于是，在处理更新事件的代码段中，您就可在由 ScrollRect 移动了各部分的窗口面积中进行绘制，请参阅 5 – 62 页的“在窗口内绘制滚动后的文稿”中的介绍。

当垂直滚动杆的滚动箭头或灰度面积中发生鼠标按下事件时，第5 - 53页程序清单5 - 18中的例程DoContentClick便调用TrackControl，并且将一个指向应用程序定义的动作过程（称为MyVerticalActionProc）的指针传送给TrackControl。对于水平滚动杆，DoContentClick调用TrackControl，并且将一个指向称为MyHorzntlActionProc的动作过程的指针传送给TrackControl。在下一节中将介绍这些动作过程。

为响应滚动箭头和灰度面积中的事件进行滚动

只要单击一下滚动箭头，您的应用程序就应该延所选的方向滚动一个单位（即一行、一个字符、一个单元或其它您的应用程序认为合适的单位）的距离。当光标位于滚动箭头上时，如果用户按住鼠标按钮，您的应用程序就应该连续滚动若干单位的距离，直至用户释放鼠标按钮或直至您的应用程序已将文稿滚动到最大距离而无法继续为止。

只要单击一次灰度面积，您的应用程序就应该以各小于一个窗口高度或窗口宽度的距离沿相应的方向滚动，以便显示出原窗口的部分（这样，就可以将新显示的资料放置在上下文中）。当光标位于灰度面积时，如果用户按住了鼠标按钮，您的应用程序就应该连续滚动若干单位的距离，直到用户释放鼠标按钮或直到您的应用程序已将文稿滚动到最大距离而无法继续为止。

当您的应用程序结束滚动时，应该使用SetControlValue来相应地移动滚动块。

根据本章前面部分的介绍，您可以使用FindControl来确定您的某一窗口的控制部件中何时发生了一个鼠标按下事件，然后，您使用TrackControl来跟随光标在控制部件中的运动，以便提供反馈，使用户可以看到光标的位置，然后当用户释放鼠标按钮时通知您的应用程序。

当一个现用滚动杆的滚动箭头或灰度面积中发生一个鼠标按下事件时，作为其结果，FindControl返回相应的部分代码。以下是用于滚动箭头和灰度面积的部分代码，以及您用来代表滚动箭头和灰度面积的常数:

常数	部分代码	控制部分
inUpButton	20	垂直滚动杆的上滚动箭头、水平滚动杆的左滚动箭头
inDownButton	21	垂直滚动杆的下滚动箭头、水平滚动杆的右滚动箭头
inPageUp	22	垂直滚动杆中滚动块以上的灰度面积、水平滚动杆中滚动块左边的灰度面积
inPageDown	23	垂直滚动杆中滚动块以下的灰度面积、水平滚动杆中滚动块右边的灰度面积

当 FindControl 返回其中一个部分代码时，您的应用程序应该立即调用 TrackControl。当光标位于滚动箭头时，只要用户按住鼠标按钮，TrackControl 便醒目显示滚动箭头，如第5－12页图5－8所示。当用户释放鼠标按钮时，TrackControl 就取消醒目显示。

对于所有其它的标准控制部件、以及滚动杆中的滚动块，只有当 TrackControl 发出报告，说明在最初发生鼠标按下事件的同一个控制部分中发生释放鼠标事件时，您的应用程序才予以响应。但是，对于滚动箭头和灰度面积，您的应用程序在得到 TrackControl的报告，说明用户已释放鼠标按钮之前，必须通过滚动文稿给予响应。当您为滚动箭头和灰度面积调用 TrackControl 时，您必须定义一个动作过程，该过程可以进行相应的滚动，直到得到 TrackControl 的报告，说明用户已释放鼠标按钮为止。

当用户释放了鼠标按钮或将光标移出滚动箭头或灰度面积时，作为其结果，TrackControl 返回原来列出的、代表控制部分的一个值。如第5　53页上的程序清单5 － 18 所示，DoControlClick 过程测试 inUpButton、inDownButton、inPageUp 和 inPageDown 的部分代码，以便确定何时在滚动箭头或灰度面积中发生鼠标按下事件。

当光标位于垂直滚动杆的滚动箭头或灰度面积时，如果用户按下或按住鼠标按钮，DoContentClick 便调用 TrackControl，并将一个指向应用程序定义的动作过程（称为 MyVerticalActionProc）的指针传送给 TrackControl。对于水平滚动杆，DoContentClick 调用 TrackControl，并且将一个指向动作过程（称为 MyVerticalActionProc）的指针传送给 TrackControl。而 TrackControl 又调用这些动作过程连续滚动，直到用户释放鼠标按钮为止。

附注

作为一种替代方法，您不必将指向您动作过程的指针在一个参数中传送到 TrackControl 中，而是使用 SetControlAction 过程将一个指向动作过程的指针存储在控制记录的 ControlAction 域中。当您向 TrackControl 传送 Pointer (-1)，而不是传送过程指针时，TrackControl 便使用在控制记录中所指向的动作过程。

程序清单5 － 19 表示两个动作过程示例：MyVerticalActionProc（用于响应垂直滚动杆的滚动箭头和灰度面积中的鼠标事件）和 MyHorzntlActionProc（用于响应水平滚动杆中的同样事件）。当 TrackControl 调用这些动作过程时，它传送一个控制部件间接指针和一个整数。（该整数代表发生鼠标事件的控制部件）MyVerticalActionProc 和 MyHorzntlActionProc 均使用常数 inUpButton、inDownButton、inPageUp 和 inPageDown 来测试"…TrackControl 传送的控制部件。

程序清单 5 – 19 用于滚动文字文稿的动作过程

```
PROCEDURE MyVerticalActionProc (control: ControlHandle; part: Integer);
VAR
    scrollDistance:         Integer;
    window:                 WindowPtr;
    myData:                 MyDocRecHnd;
BEGIN
    IF part <> 0 THEN
    BEGIN
        window := control^^.contrlOwner;     {get the control's window}
        myData := MyDocRecHnd(GetWRefCon(window));
        HLock(Handle(myData));
        CASE part OF
            inUpButton, inDownButton:  {get one line to scroll}
            scrollDistance := 1;
            inPageUp, inPageDown:      {get the window's height}
            BEGIN
                scrollDistance :=(myData^^.editRec^^.viewRect.bottom -
                              myData^^.editRec^^.viewRect.top)
                              DIV myData^^.editRec^^.lineHeight;
                {subtract 1 line so user sees part of previous window}
                scrollDistance := scrollDistance - 1;
            END;
        END;     {of part CASE}
        IF (part = inDownButton) OR (part = inPageDown) THEN
            scrollDistance := -scrollDistance;
        MyMoveScrollBox(control, scrollDistance);
        IF scrollDistance <> 0 THEN          {scroll by line or by window}
            TEPinScroll(0, scrollDistance * myData^^.editRec^^.lineHeight,
                        myData^^.editRec);
        HUnLock(Handle(myData));
    END;
END; {of MyVerticalActionProc}

PROCEDURE MyHorzntlActionProc (control: ControlHandle; part: Integer);
VAR
    scrollDistance:         Integer;
    window:                 WindowPtr;
    myData:                 MyDocRecHnd;
BEGIN
    IF part <> 0 THEN
    BEGIN
        window := control^^.contrlOwner;     {get the control's window}
```

```
myData := MyDocRecHnd(GetWRefCon(window));
HLock(Handle(myData));
CASE part OF
     inUpButton, inDownButton:  {get a few pixels}
                   scrollDistance := kButtonScroll;
          inPageUp, inPageDown:  {get a window's width}
                   scrollDistance := myData^^.editRec^^.viewRect.right -
                                    myData^^.editRec^^.viewRect.left;
        END;      {of part CASE}
        IF (part = inDownButton) OR (part = inPageDown) THEN
            scrollDistance := -scrollDistance;
        MyMoveScrollBox(control, scrollDistance);
        IF scrollDistance <> 0 THEN
            TEPinScroll(scrollDistance, 0, myData^^.editRec);
        HUnLock(Handle(myData));
     END;
END; {of MyHorzntlActionProc}
```

每个动作过程在开始时首先要确定一个适当的滚动距离。对于垂直滚动杆中的滚动箭头，MyVerticalActionProc 将滚动距离定义成一行。对于垂直滚动杆中的灰度面积，MyVerticalActionProc通过将窗口高度除以行高度可确定滚动距离的行数;窗口的高度等于显示矩形的顶部坐标减去其底部坐标（在编辑记录中定义）。然后 MyVerticalActionProc将此距离减去1，这样，当光标位于灰度面积时，如果用户按下鼠标按钮，MyVerticalActionProc 滚动的行数就比窗口中显示的总行数少一行。

MyVerticalActionProc过程随后将这些行距离乘以行高度以得到像素距离，使之能够传送到TEPinScroll的参数中。另外，当下面滚动箭头或滚动块以下的灰度面积中发生鼠标按下事件时，MyVerticalActionProc 将这些距离转变成负数值。

对于水平滚动杆中滚动箭头的滚动距离，MyHorzntlActionProc 使用预先确定的像素距离－－大约是文稿的平均字符宽度。对于灰度面积的滚动距离，MyHorzntlActionProc 使用窗口宽度（将显示矩形的右坐标减去其左坐标即可得到）。当右滚动箭头或滚动块右边的灰度面积中发生鼠标按下事件时，MyHorzntlActionProc 例程便将这些距离转变成负数值。

在调用应用程序定义的例程 MyMoveScrollBox 移动滚动块以后，两个动作过程的使用 TEPinScroll 按照该滚动距离移动窗口中显示的文字。（在此实例中，应用程序 SurfWriter 滚动的是存储在 TextEdit 编辑记录中的简单、单一字形的文字文稿。关于使用较通用的 QuickDraw 滚动例程 ScrollRect 的讨论，请参阅 5－53 页“为响应滚动块中的事件而滚动屏幕”一节。）

TEPinScroll 过程可以自动地创建一个更新区域并请求一个更新事件。应用程序 SurfWriter 在其窗口更新代码段中使用 TEUpdate 过程来绘制更新区域中的文字，如 5 – 65 页程序清单 5 – 23 所示。

动作过程以指定的距离继续反复地移动文字，直到用户释放鼠标按钮并且 TrackControl 结束为止。如果滚动到没有可滚动的区域，TEPinScroll 便自动地停止滚动，这类似于您执行您自己的滚动例程时您的应用程序的情形。

从程序清单 5 – 20 中可以看到当动滚动文稿时，应用程序定义的过程 MyMoveScrollBox 如何使用 GetControlValue、GetControlMaximum 和 SetControlValue 过程将滚动块移动适当的距离。MyMoveScrollBox 过程使用 GetControlMaximum 来确定最大滚动距离，使用 GetControlValue 来确定滚动块的当前设定状态，使用 SetControlValue 来指派新的设定状态并移动滚动块。5 – 37 页的“确定和更改控制设定状态”中介绍了如何使用 SetControlMaximum 和 SetControlValue 例程；在 5 – 104 页中详细介绍了 GetControlMaximum。

程序清单 5 – 20 用动作过程移动滚动块

```
PROCEDURE MyMoveScrollBox (control: ControlHandle;
                           scrollDistance: Integer);
VAR
    oldSetting, setting, max:     Integer;
BEGIN
    oldSetting := GetControlValue(control); {get last setting}
    max := GetControlMaximum(control);  {get maximum setting}
{subtract action procs' scroll amount from last setting to get new setting}
    setting := oldSetting - scrollDistance;
    IF setting < 0 THEN
        setting := 0
    ELSE IF setting > max THEN
        setting := max;
    SetControlValue(control, setting); {assign new current setting}
END; {of MyMoveScrollBox}
```

前两节介绍了如何移动显示于窗口中的各部份，下一节将介绍如何在更新区域中进行绘制。

在窗口内绘制滚动后的文稿

在前面两节中，已经介绍了如何使用 QuickDraw 的过程 ScrollRect 和 TextEdit的过程TEPinScroll来响应用户对滚动杆五个部分中任一部分的操纵。在使用这些过程或您自己的例程来移动窗口中的各个位以后,您的应用程序必须在更新区域中进行绘制。为此，您通常应使用您自己的窗口更新代码段。

在本章前面所示的实例中用到的InvalRect和TEPinScroll均可以创建更新区域，从而产生更新事件。根据本书“窗口管理器”和“事件管理器”两章的介绍，当您的应用程序接收到更新事件时，应该在您窗口的更新区域中进行绘制。如果您创建并使用您自己的滚动例程，而不使用 ScrollRect 或 TEPinScroll，您就应该保证它能够生成一个更新事件，或者保证它可以显式调用您自己的窗口更新例程。

程序清单 5 - 21 表示一个应用程序定义的例程 DoUpdate，只要应用程序 SurfWriter 接收到一个更新事件，它便调用此例程。在此过程中，应用程序测试两个不同类型的窗口：包含图形对象的窗口和包含用 TextEdit 例程创建的文字的窗口。

程序清单 5 - 21 一个应用程序定义的更新例程

```
PROCEDURE DoUpdate (window: WindowPtr);
VAR
     windowType: Integer;
BEGIN
     windowType := MyGetWindowType(window);
     CASE windowType OF
     kMyGraphicsWindow:     {window containing graphics objects}
          BEGIN
               BeginUpdate(window);
               MyDrawGraphicsWindow(window);
               EndUpdate(window);
          END;      {of updating graphics windows}
     kMyDocWindow:          {window containing TextEdit text}
          BEGIN
               BeginUpdate(window);
               MyDrawWindow(window);
               EndUpdate(window);
          END;      {of updating TextEdit document windows}
     {handle other window types——modeless dialogs, etc.——here}
     END;      {of windowType CASE}
END;      {of DoUpdate}
```

在此实例中，当需要更新的窗口类型是kMyGraphicsWindow时，DoUpdate使用另一个称为MyDrawGraphicsWindow的应用程序定义的例程。当需要更新的窗口类型是kMyDocWindow时，DoUpdate使用另一个称为MyDrawWindow的应用程序定义的例程。程序清单5－22表示例程MyDrawGraphicsWindow，5－65页的程序清单5－23表示例程MyDrawWindow。

在绘制进行到窗口的滚动离开部分以前，这两个例程均需要使用QuickDraw、窗口管理器和控制管理器的例程来更新窗口。（5－29页的“更新控制部件”中介绍了UpdateControls过程；请参阅本书的“窗口管理器”一章中关于如何使用其余这些例程来更新窗口的详细说明。）

程序清单5－22 刷新包含有图形对象的窗口

```
PROCEDURE MyDrawGraphicsWindow (window: WindowPtr);
VAR
     myData:              MyDocRecHnd;
     i:              Integer;
BEGIN
     SetPort(window);
     myData := MyDocRecHnd(GetWRefCon(window));
     HLock(Handle(myData));
     WITH window^ DO
          BEGIN
               EraseRect(portRect);
               UpdateControls(window, visRgn);
               DrawGrowIcon(window);
               SetOrigin(GetControlValue(myData^^.hScrollBar),
                         GetControlValue(myData^^.vScrollBar));
               i := 1;
               WHILE i <= myData^^.numObjects DO
               DrawMyObjects(portRect, myData^^.numObjects[i]);
                    i := i + 1;
               END; {of WHILE}
               SetOrigin(0, 0);
          END;
     HUnLock(Handle(myData));
END;      {of MyDrawGraphicsWindow}
```

MyDrawGraphicsWindow例程使用QuickDraw的过程SetOrigin来更改窗口起始位置，改变量等于滚动杆的设定状态，这样文稿的左上角就位于窗口局部坐标系统的（0，0）上。示例应用程序SurfWriter执行此步骤，从而使它自己的绘制例程可以在正确的窗口区域中进行绘制。

请注意，MyDrawGraphicsWindow仅在调用必需的窗口管理器例程和控制管理器例程以后才调用SetOrigin，因为窗口管理器和控制管理器总是要求窗口起始位置是在（0，0）。

通过使用SetOrigin来更改窗口起始位置，MyDrawGraphicsWindow可以把文稿中的对象看作是定位在一个文稿左上角总是位于（0，0）的坐标系统中。然后，MyDrawGraphicsWindow调用它自己的另外几个例程中的DrawMyObjects，以便在窗口中绘制已存储在其文稿记录中的对象。

MyDrawGraphicsWindow在执行它自己所有的窗口绘制以后，再次使用Set-Origin过程—这次是要将窗口起始位置恢复到（0，0），以便以后的窗口管理器例程和控制管理器例程能够绘制到正确的窗口区域中。

前面的图5－16至图5－20辅助说明了如何使用SetOrigin来偏移窗口的坐标系统，以便把您文稿中的对象看作是固定在文稿本身的坐标空间中。但是，您的应用程序并不是一定要依赖 SetOrigin。您的应用程序可以将窗口的坐标系统保持固定不变，而用等于滚动杆的设定状态的改变量来偏移文稿中的项目。如果您倾向于使用这种方法，那么就可以使用在《Inside Macintosh 绘图》*(Inside Macintosh: Imaging)* 中介绍的QuickDraw的过程 OffsetRect、OffsetRgn、SubPt和AddPt。

附注

SetOrigin过程不移动窗口的剪贴区域。如果您在窗口中使用剪贴区域，就可以使用QuickDraw的过程GetClip，以便在您首次调用SetOrigin之后立即存储您的剪贴区域。在调用您自己的窗口绘制例程之前，请使用QuickDraw的过程ClipRect来定义新的剪贴区域，这样可以避免绘制时侵入滚动杆区域等情况。在调用您自己的窗口绘制例程之后，请使用QuickDraw过程ClipRect来还原初始的剪贴区域。然后您可以再次调用SetOrigin，将窗口起始位置还原到（0，0），而且保持初始的剪贴区域完整无损。请参阅《Inside Macintosh 绘图》*(Inside Macintosh: Imaging)* 中关于剪贴区域和这些QuickDraw例程的详细说明。

本章前面所举的实例曾介绍一个应用程序，它利用一个TextEdit编辑记录来存储用户创建的信息。出于简化文字处理的需要，TextEdit提供的许多例程可以简化您的工作；例如，TEPinSeroll过程（在5－53页程序清单5－18和5－59页程序清单5－19中曾经用到）根据您的应用程序指定象素数量将存储在一个编辑记录中文字的显示矩形复位。然后，TEPinScroll过程生成一个窗口更新事件。TextEdit的过程TEUpdate随后在一个应用程序的更新例程中被调用，以便绘制滚动后窗口的更新区域。

程序清单5－23表示一个应用程序定义的过程MyDrawWindow，该过程使用TEUpdate来更新kMyDocWindow类型窗口中的文字。在进行窗口更新时，TEUpdate过程可管理所有必要的坐标移动，因此，MyDrawWindow不必象它使用ScrollRect时一样调用SetOrigin。

程序清单 5 － 23 在滚动一个 TextEdit 编辑记录后刷新窗口

```
PROCEDURE MyDrawWindow (window: WindowPtr);
VAR
     myData: MyDocRecHnd;
BEGIN
     SetPort(window);
     myData := MyDocRecHnd(GetWRefCon(window));
     HLock(Handle(myData));
     WITH window^ DO
          BEGIN
               EraseRect(portRect);
               UpdateControls(window, visRgn);
               DrawGrowIcon(window);
               TEUpdate(portRect, myData^^.editRec);
          END;
     HUnLock(Handle(myData));
END;      {of MyDrawWindow}
```

移动滚动杆和调整滚动杆的大小

正如前面 5 － 21 页“创建滚动杆”介绍的那样，起初您的应用程序通过在控制资源中或NewControl的参数中指定一个矩形来定义滚动杆在窗口中的位置和滚动杆的大小。但您的应用程序在响应用户对您的窗口进行大小调整时，必须能够动态地调整滚动杆大小和移动滚动杆。

本书中的“窗口管理器”一章，介绍了当您的应用程序打开窗口时如何确定窗口的大小，以及如何调整它们的大小（例如在响应用户拖拽尺寸格或单击缩放格时）。本节介绍怎样移动滚动杆并调整它的大小，使它们正好处于您的窗口的底部和右边缘。

当调整您的窗口的大小时，您的应用程序应该执行下列步骤来调整每一个滚动杆。

1. 调整窗口大小。

2. 使用 HideControl 过程来隐藏每个滚动杆。

3. 使用 MoveControl 过程把垂直滚动杆移至窗口的右边缘，又用 MoveControl 过程把水平滚动杆移至窗口的底边缘。

4. 使用 SizeControl 过程拉长或切短每个滚动杆，使得每个滚动杆都伸展到窗口右下角的尺寸格。

5. 重新计算滚动杆的最大设定值，使用 SetControlMaximum 来更新设定状态并相应地重新绘制滚动杆。（记住，从文稿的长度或宽度中减去其窗口的长度或宽度您就得到一个滚动杆的最大设定值。）

6. 调用 ShowControl 过程使每个滚动杆在它的新位置可视。

图 5 － 21 说明在一个调整了大小的窗口中如何移动滚动杆和调整其大小；如果您的应用程序不使用 HideControl 过程，用户将会看见每一个步骤的发生情况。

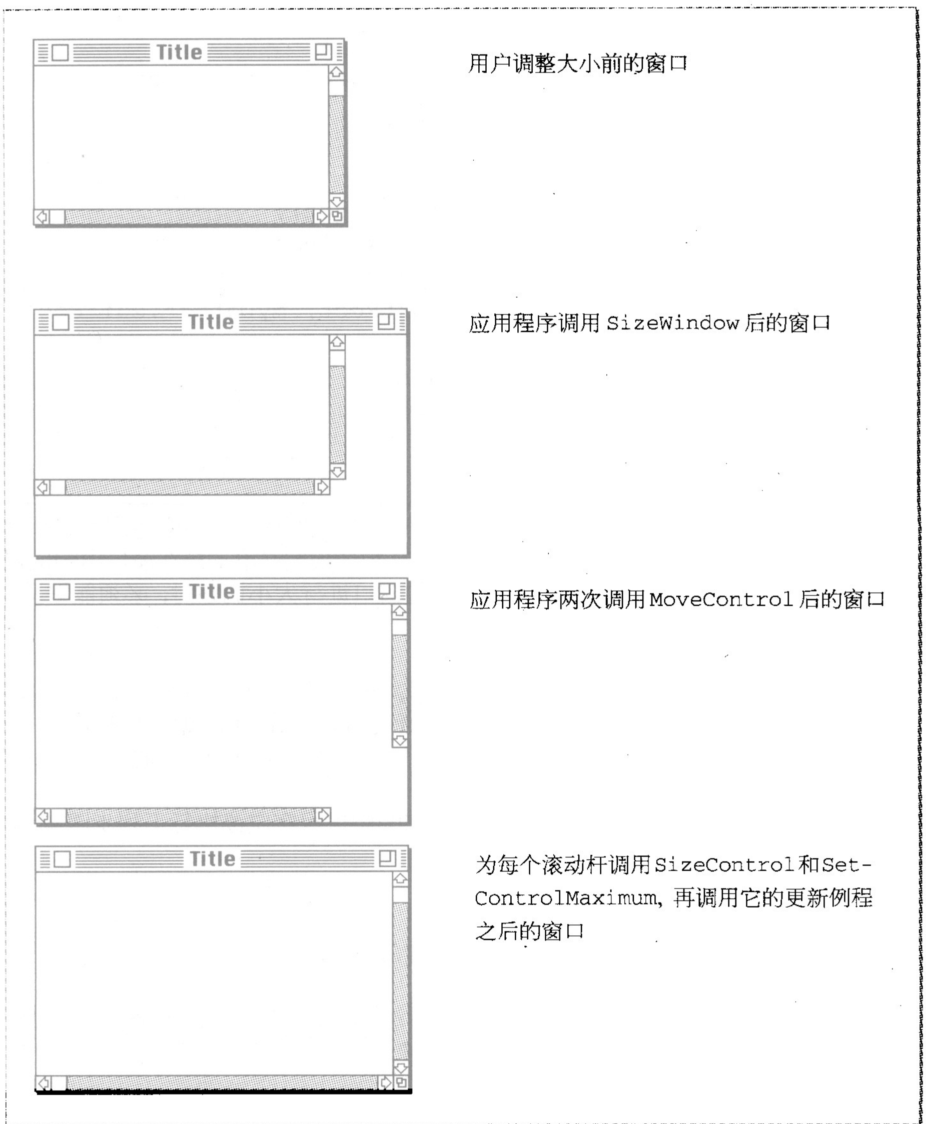


图 5 － 21　　移动滚动杆和调整它的大小

5 - 39页的程序清单5 - 14显示一个应用程序定义的例程MyAdjustScrollBars，当用户打开一个新窗口，在窗口中打开一个已存在的文稿，或调整窗口大小时，就调用MyAdjustScrollBars。

当MyAdjustScrollBars创建窗口时，它在一个文稿记录中存储每个滚动杆的间接指针。通过间接引用文稿记录中合适的域，MyAdjustScrollBars将垂直和水平滚动杆的间接指针传递给HideControl过程，这个过程使滚动杆变成不可视。MyadjustScrollBars直到它完成操纵滚动杆才使滚动杆可视，这就保证了用户看不到滚动杆在屏幕不同位置闪烁。

当MyAdjustScrollBars需要调整任意一个滚动杆的大小或位置时，它将调用另一个应用程序定义的例程MyAdjustScrollSizes，程序清单5 - 24给出了这个例程。

程序清单 5 - 24 改变窗口滚动杆的大小和位置

```
CONST
    kScrollbarWidth = 16;                       {conventional width}
    kScrollbarAdjust = kScrollbarWidth - 1;  {to align with window frame}
    kScrollTweek = 2;                       {to align scroll bars with size box}

PROCEDURE MyAdjustScrollSizes (window: WindowPtr);
VAR
    teRect:                             Rect;
    myData:                             MyDocRecHnd;
    teTop, teRight, teBottom,teLeft:    Integer;
BEGIN
    MyGetTERect(window, teRect);    {calculate the teRect based on the }
                                    { portRect, adjusted for the scroll bars}
    myData := MyDocRecHnd(GetWRefCon(window));
    HLock(Handle(myData));
    WITH window^.portRect DO
    BEGIN
        teTop := top;
        teRight := right;
        teBottom := bottom;
        teLeft := left;
    END;
    WITH myData^^ DO
    BEGIN
        editRec^^.viewRect := teRect;   {set the viewRect}
        MyAdjustViewRect(editRec);      {snap to nearest line}
        {move the controls to match the new window size}
        MoveControl(vScrollBar, teRight - kScrollbarAdjust, -1);
```

```
            SizeControl(vScrollBar, kScrollbarWidth, (teBottom - teTop) -
                    (kScrollbarAdjust - kScrollTweek));
            MoveControl(hScrollBar, -1, teBottom - kScrollbarAdjust);
            SizeControl(hScrollBar, (teRight - teLeft) -
                (kScrollbarAdjust - kScrollTweek), kScrollbarWidth);
        END;
        HUnLock(Handle(myData));
    END; {of MyAdjustScrollSizes}
```

MyAdjustScrollSizes 例程使用窗口内容区域的界定矩形（存储在窗口记录的 PortRect 域）来确定窗口的大小。若要将滚动杆移至窗口的边缘，那么 MyAdjustScrollSizes 需要使用 MoveControl 过程。

MoveControl 过程需要 3 个参数：被移动的控制部件的间接指针，控制部件矩形左上角的新位置的水平坐标（控制部件窗口的局部坐标），以及这个新位置的垂直坐标。MoveControl 过程把控制部件移到这个新位置，并更改在控制部件的控制记录的 controlRect 域中指定的这个矩形。

在程序清单 5 - 24 中，MyAdjustScrollSizes 把滚动杆的间接指针传递给 MoveControl。（SurfWriter 示例应用程序在其窗口的文稿记录中存储间接指针。）

图 5 - 22 给出了一个滚动杆在移至其调整大小后的窗口内的新位置之前的位置。

MyAdjustScrollSizes从窗口的右坐标中减去15就得出垂直滚动杆左上角新的水平（即左）坐标。如图 5 - 23 所示，这样就将 16 像素宽滚动杆的右边缘直接叠加在窗口右边 1 像素宽的框架上。

在 5 - 67 页的程序清单 5 - 24 中，MyAdjustScrollSizes 指定 - 1 作为垂直滚动杆左上角的垂直（即顶部）坐标。如图 5 - 23 所示，这样就将滚动杆的顶部边缘直接叠加在标题栏 1 像素宽的底线上。（标题栏底线在窗口的局部坐标系统中的垂直坐标是 - 1。）

MyAdjustScrollSizes 例程指定 - 1 作为水平滚动杆左上角的水平坐标；这样就将水平滚动杆的左边缘直接叠加在 1 像素宽的窗口框架上。（窗口的左边框架在窗口局部坐标系统中的水平坐标值为 - 1。）

若要使您的滚动杆能在窗口框架中很好地吻合，您就应该将垂直滚动杆的顶部坐标设定为 - 1，将水平滚动杆的左坐标设定为 - 1。但如果您的应用程序使用窗口滚动杆的一部分来对应尺寸格，用于显示信息或额外的控制部件，您就不应作这样的设定。例如，您可以选择在窗口左下角显示文稿的当前页号。在这种情况下，应指定一个左坐标使得水平滚动杆不遮住这个区域。

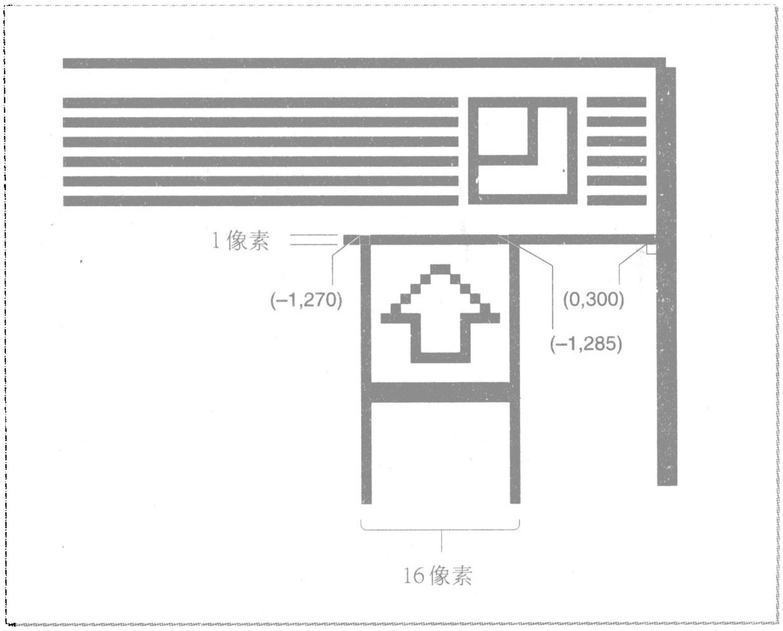


图 5 – 22　　调整大小后的窗口内的滚动杆，此时应用程序还没有移动该滚动杆

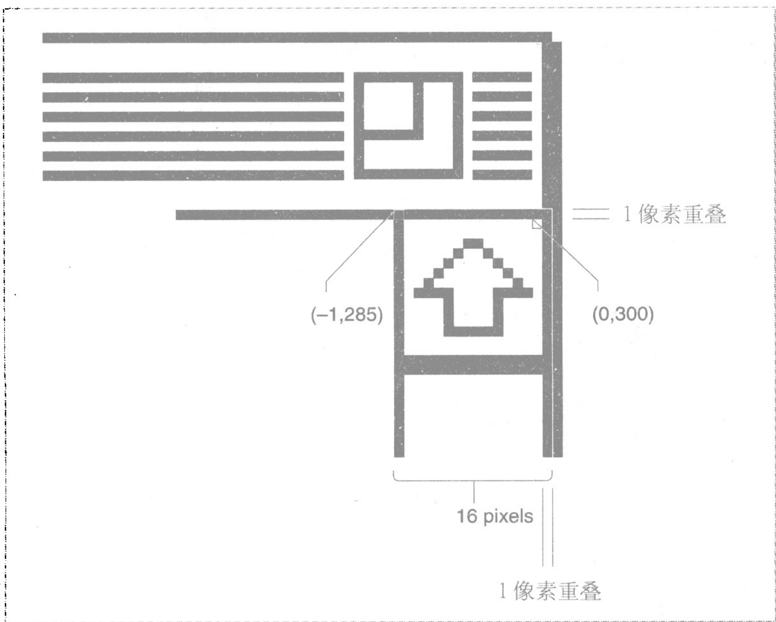


图 5 – 23　　应用程序移动其左上点后的垂直滚动杆

请参阅《Macintosh用户界面设计原则》*(Macintosh Human Interface Guidelines)* 中关于正确使用滚动杆区域，把它用于项目而非滚动杆的讨论。

MyAdjustScrollSizes从窗口底部坐标中减去15得出水平滚动杆左上角新的垂直坐标；这样就把滚动杆底边缘直接叠加在窗口底部框架上。

MoveControl过程移动滚动杆左上角使得它处于窗口框架内合适的位置。若要实现垂直滚动杆与窗口的高度吻合，实现水平滚动杆与窗口的宽度吻合，MyAdjustScrollSizes则应使用SizeControl过程。

SizeControl过程需要3个参数：被调整大小的控制部件的间接指针，以像素为单位控制部件的宽度，以及以像素为单位控制部件的高度。当调整垂直滚动杆大小时，您可以调整它的高度；当调整水平滚动杆大小时，您可以调整它的宽度。

当使用SizeControl来调整垂直滚动杆时，MyAdjustScrollSizes传递一个代表16像素的常数作为垂直滚动杆的宽度，这是常规大小。

若要确定这个滚动杆的正确高度，MyAdjustScrollSizes首先从窗口矩形底部的坐标中减去其顶部的坐标，得出窗口的高度。然后MyAdjustScrollSizes从这个窗口的高度中再减去13个像素，并把结果传递给SizeControl作为垂直滚动杆的高度。

MyAdjustScrollSizes例程从窗口高度中减去13个像素是为了给16像素高的尺寸格（在窗口底部）留下空间，其中有3个像素的差是因为有3次1像素重叠：一次在窗口框架的顶部，一次在尺寸格的顶部，还有一次在尺寸格的底部。

当使用SizeControl来调整水平滚动杆时，MyAdjustScrollSizes传递一个代表16个像素的常数－－这是水平滚动杆的常规高度。若要确定这个滚动杆的正确宽度，MyAdjustScrollSizes首先从窗口矩形的右坐标中减去其左坐标，得出窗口的宽度。MyAdjustScrollSizes再从这个窗口宽度中减去13个像素以便为尺寸格留出地方（正如在确定垂直滚动杆高度时处理的一样）。

当MyAdjustScrollSizes结束时，它返回至MyAdjustScrollBars，MyAdjustScrollSizes然后使用另外一个它自己的例程MyAdjustScrollValues。MyAdjustScrollValues又调用MyAdjustHV（在5－41页的程序清单5－16中），MyAdjustHV重新计算滚动杆的最大设定，使用SetControlMaximum来更新最大设定并相应地刷新滚动杆。

当MyAdjustHV结束时，它最终返回至SurfWriter应用程序的MyAdjustScrollBars过程，这个过程再调用ShowControl过程使最新调整的滚动杆重新可视。

定义您自己的控制部件定义函数

控制管理器允许您实现标准控制部件（按钮、复选框、选项按钮、弹出式菜单和滚动杆）以外的其它控制部件。若要实现非标准控制部件，您必须定义您自己的控制部件定义函数。通常，您可能需要实现的控制部件类型仅仅是滑块或拨号，它们与滚动杆相似，在其中它们以图形的方式展示用户可设定的值的范围。像滚动杆有滚动块一样，您的滑块和拨号应该有关于设定值和表明当前设定状态的指示符。

拨号和滑块显示某些东西的大小、幅度或位置。一般以一些伪摸拟的形式显示，例如滑动开关的位置、Tick 表的读数或仪表指针的角度；也可以用数字的方式显示设定状态。用户应该能够通过拖拽控制部件的指示符改变它的设定状态。

图5－24给出了由应用程序定义的控制部件定义函数支持的一个控制部件。这个控制部件可能用来重放一段语音或一段 QuickTime 活动影像。应用程序希望定义这个控制部件，使得用户单击左边的控制部件时它能以正常的速度播放语音或移动影像。应用程序沿着滑块使用指示符显示当前正在播放的是语音或移动的那部分影像序列。应用程序也允许用户通过拖拽指示符在序列中快速向前移动或后退。最后，应用程序希望在最右端定义两个控制部件，使得用户单击它们时，它们能分别地“快倒”和“快进”。

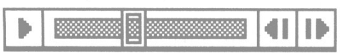


图5－24 自定控制部件

附注

当您设计拨号或滑块时，一定要包括有意义的标签，以便向用户表明指示符的范围和方向。

您可能会临时地把滚动杆用于这种目的，而自己不创建这样的控制部件。请不要这样做。因为滚动杆只能用于在窗口中滚动，如果把滚动杆用于其它目的将会破害 Macintosh 界面的一致性。

若要定义您自己的非标准控制部件，您必须编写一个控制部件定义函数，把它作为一个 'CDEF' 类型的资源编译器，并存储在您的资源文件中。[若想了解关于创建资源的更多的信息，请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 的“资源管理器”一章]。

当您使用控制管理器例程时，这些例程又根据需要调用您的控制部件定义函数。例如，若要使图5 – 24中的控制部件能够正常地工作，它的控制部件定义函数必须能够实现以下功能：

- 绘制控制部件——包括其指示符的重定位，当鼠标事件在它的控制部件有效区域中发生时相应地醒目显示这些控制部件
- 确定在一控制部分中什么时候发生鼠标按下事件的时间
- 计算控制部件及其指示符的区域
- 移动指示符并用新设定状态来更新控制记录

您也可以使用控制部件定义函数来修改或扩展某些控制管理器行为。例如，您可实现您自己的拖拽指示符的方式，您还可执行您自己的控制部件初始化类型。

若想了解关于编写控制部件定义函数的详细介绍，请参阅5 – 109页的“定义您自己的控制部件定义函数。”

控制管理器参考资料

这部分介绍控制管理器特定的数据结构、例程和资源。

“数据结构”这一节介绍了控制记录的数据结构，辅助控制记录，弹出式菜单专用数据记录，以及控制部件色彩表记录。“控制管理器例程”这一节介绍了控制管理器的例程，它们用于创建控制部件，绘制控制部件，处理控制部件中的鼠标事件，更改控制设定状态和显示、确定控制设定状态，以及取消控制部件。“应用程序定义的例程”这一节介绍了控制部件定义函数，当定义您自己的控制部件时您需要提供控制部件定义函数。“应用程序定义的例程”这一节还介绍了动作过程。这个过程定义，只要当光标在一个控制部件中用户按住鼠标按钮，就不断地重复执行一个动作。“资源”这一节介绍了控制资源和控制部件色彩表资源。

数据结构

本节介绍控制记录、辅助控制记录、弹出式菜单专用数据记录以及控制部件色彩表记录等内容。

您的应用程序并不特别地创建控制记录、辅助控制记录或弹出式菜单的专用数据记录；相反，您的应用程序只需创建一些必要的资源并使用合适的控制管理器例程。控制管理器将根据需要创建这些记录。

您可使用控制管理器例程来更改控制记录中的数值，您自己也可存取和修改它的域。您一般不更改辅助控制记录中的值。但在控制记录和辅助控制记录中都有这样一些域：您的应用程序可在其中存储您认为合适的信息。

弹出式菜单控制部件的控制记录的ContrlData域中包含弹出式菜单专用数据记录的间接指针。弹出式菜单专用数据记录中包含相关菜单的菜单间接指针和菜单标识符。通过间接引用控制记录的 ControlData 域，您可获得与弹出式菜单相关的菜单的菜单间接指针和菜单标识符。

只有当您要在应用程序运行时创建的控制部件中使用非标准的色彩时，您才使用控制部件色彩表记录。您的应用程序最好不要创建控制部件色彩表记录，因为您应该使用系统的系统设定色彩以保证应用程序界面的一致性。

控制记录

当您创建控制部件时，控制管理器把您指定的信息（在控制资源或NewControl函数的参数中）加到控制记录中，控制记录是一个ControlRecord类型的数据结构。您用来创建控制部件的控制管理器函数GetNewControl和NewControl，返回最新分配的控制记录的间接指针，这以后，您的应用程序一般通过这个间接指针访问控制部件，因为大多数其它控制管理器例程把控制部件间接指针作为它们的第一个参数。

您可使用控制管理器例程来确定和更改控制记录中的几个值，您也可自己存取或修改控制记录的域。

```
TYPE ControlRecord =
PACKED RECORD
     nextControl:      ControlHandle;{next control}
     contrlOwner:      WindowPtr;     {control's window}
     contrlRect:       Rect;          {rectangle}
     contrlVis:        Byte;          {255 if visible}
     contrlHilite:     Byte;          {highlight state}
     contrlValue:      Integer;       {control's current setting}
     contrlMin:        Integer;       {control's minimum setting}
     contrlMax:        Integer;       {control's maximum setting}
     contrlDefProc:    Handle;        {control definition function}
     contrlData:       Handle;        {data used by contrlDefProc}
     contrlAction:     ProcPtr;       {action procedure}
     contrlRfCon:      LongInt;       {control's reference value}
     contrlTitle:      Str255;        {control's title}
END;
```

地域说明

nextControl 与这个控制部件的窗口相关的下一个控制部件的间接指针。属于一个给定窗口的所有控制部件均保持在一个链接列表中，此列表的起始端在窗口记录的 controlList 域中，通过单个控制记录 nextControl 域链接在一起。这个列表的结尾以 NIL 作标记；一旦创建了新的控制部件，他们就被添加到列表的开始端。

controlOwner 这个控制部件所属窗口的指针。

ContrlRect 在控制部件的窗口的局部坐标中，完全包围控制部件的矩形。您可使用MoveControl和SizeControl过程来改变存储在这个域中的矩形。

contrlVis 控制部件的不可视/可视状态。当这个域的值是0时，控制管理器不绘制控制功能（它的状态是不可视）；当这个域的值是255时，控制管理器绘制这个控制部件（它的状态是可视）。注意，尽管一个控制部件是可视的，它有时也可能被一个叠加窗口或一些其它对象遮住。您可使用HideControl过程把这个域从可视变成不可视，您也可使用 ShowControl 过程把这个域从不可视变成可视。

contrlHilite 指定是否显示控制部件以及如何显示，指出它是有效的还是无效的，如果是当前活动的话它指出它是现用还是失活的，如果是现用的话它是否被选定。值为 0 表示没被选定的一个现用控制部件。从 1 到 253之间的一个数值表示一个部分代码，指定醒目显示（有效）控制部件的某部分，表明光标在这个部分中时用户正按下鼠标按钮。如值为 255 则表示控制部件状态为无效状态并相应地对它重新绘制。HiliteControl 过程让您更改这个域的值。

contrlValue 控制部件的当前设定状态。对于按钮、复选框和选项按钮来说，0 意味着控制部件是关闭的；1 意味着控制部件是打开的。对于滚动杆和其它滑块来说，contrlValue可以是contrlMin和contrlMax域指定的范围内的任何数值。对于弹出式菜单来说，这个数值是用户所选的菜单命令项目的项目号码；如果用户没有选择菜单命令项目，它就是第一个菜单命令项目的项目号码。对于其它控制部件，您可以随心所欲地使用这个域。您可使用 GetControlValue 函数来确定这个域的值，您还可使用 SetControlValue 过程来改变这个域的值。

ControlMin 控制部件的最小可能设定。对于开关控制部件（如复选框和选项按钮）来说这个值应该是 0（表示控制部件处于关闭状态）。对于那些不保留设定状态的控制部件（如按钮），这个值应该为0。对于弹出式菜单，控制管理器把这个域设定成 1。对于其它的控制部件，您可随心所欲地使用这个域。您可使用GetControlMinimum函数来确定这个域的值，您还可使用 SetControlMinimum 过程来改变这个域的值。

Contrlmax 控制部件的最大可能设定状态。对于开关控制部件（如复选框和选项按钮）来说这个值应是1（表示控制部件处于打开状态）。对于滚动杆和其它滑块来说，它可以是任何合适的最大值。当您使滚动杆的最大设定状态等于它的最小设定状态时，控制部件定义函数将自动地使滚动杆成为无效。当您使最大设定状态超出最小设定状态时，控制部件定义函数使滚动杆重新成为有效。对于那些不保留设定状态的控制部件（如按钮），这个值应该为1。对于弹出式菜单，控制管理器把这个值设定成菜单中项目的数目。对于其它控制部件来说，您可随心所欲地使用这个域。您可使用 GetControlMaximum 函数来确定这个域的值，您还可使用 SetControlMaxium 过程来改变这个域的值。

contrlDefProc 这种类型控制部件的控制部件定义函数间接指针。当您创建一个控制部件时，您通过一个控制部件定义标识符标识它的类型，这个控制部件定义标识 符被转换成控制部件定义函数的间接指针并存储在这个域中。这以后，控制管理器就使用这个间接指针存取定义函数；您不应该、也始终没有必要直接访问这个域。

附注

在以24位摸式运行的系统中，contrlDefProc 域的高位字节包含控制管理器从控制部件定义标识符中获取的变异值。

contrlData 保留给控制部件定义函数使用，一般用来存储一个特别控制类型特定的附加信息。例如，滚动杆的控制部件定义函数把这个域用作包含滚动块的区域的间接指针。（如果所需的附加信息不超过4个字节，定义函数就可能直接在contrlData域中存储这个信息，而不使用间接指针。）弹出式菜单的控制部件定义函数使用这个域存储一个弹出式专用数据记录，在5－77页中有相关的介绍。

contrlAction 控制部件的动作过程指针（如果有的话）。TrackControl 函数调用这个过程响应用户拖拽控制部件，这个过程可作出的响应就是只要用户按住鼠标按钮就重复地执行一些动作。请参阅5－90页对TrackControl的说明中关于动作过程更多的信息。您可使用GetControlAction函数来确定这个域的当前值，还可用SetControlAction过程来改变它。

ContrlRfCon 控制部件的参考值，您的应用程序将它用于任何目的。您可用 GetControlReference 函数来确定这个域的当前值，用 SetControlReference 过程来更改它。

ContrlTitle 控制部件标题（如果有的话）。您可使用GetControlTitle过程来确定这个域的当前值，用 SetControlTitle 过程来更改它。

辅助控制记录

为了在以32一位模式运行（用户可用存储器控制板来选定）的系统中绘制所有的控制部件，为了绘制那些使用系统系统设定以外的彩色的控制部件，控制管理器创建并保留一个链接的辅助控制记录列表，这个列表的起始端在全局变量AuxCtlHead中(所有窗口的所有控制部件仅有一个全局列表，没有针对每个窗口的单独的列表。相反，每个窗口记录有一个它自己的控制部件列表的间接指针。)

辅助控制记录是一个AuxCtlRec类型的数据结构。您的应用程序不创建、一般也不应该操纵控制部件的辅助控制记录，您应让控制管理器来创建和操纵辅助控制记录。若要创建使用系统系统设定色彩以外的彩色的控制部件，使用SetControlColor过程（5 - 101页有介绍）或创建一个控制部件色彩表资源（5 - 121页有介绍）并让控制管理器创建必要的辅助控制记录。但在辅助控制记录中有一个域，您可把它用来存储您认为合适的信息；若要获取控制部件的辅助控制记录间接指针，您可使用GetAuxiliaryControlRecord函数（5 - 107页有介绍）。

每个辅助控制记录都可重定位并驻留在您的应用程序堆中。下面介绍一个辅助控制记录是怎样定义的:

```
TYPE AuxCtlRec =
RECORD
     acNext:   AuxCtlHandle;       {handle to next AuxCtlRec}
     acOwner: ControlHandle;       {handle to this record's control}
     acCTable: CCTabHandle;        {handle to control color table }
                                   { record}
     acFlags:       Integer;       {reserved}
     acReserved:   LongInt;        {reserved for future use}
     acRefCon:     LongInt;        {for use by application}
END;
```

地域说明

acNext　辅助控制列表中下一个记录的间接指针。
acOwner　这个辅助记录所属的控制部件的间接指针；当作一个ID域使用。
acCTable　控制部件色彩表记录间接指针。(在5 - 77页中有关于控制部件色彩表记录的介绍。)
acFlags　保留给控制管理器使用
acReserved 保留给将来扩展时用。
acRefCon　一个参考值，您的应用程序可把它用于任何目的。

在使用32一位模式的系统中，每一个控制部件有它自己的辅助记录。如果您的应用程序不使用SetControlColor过程，也不创建控制部件色彩表资源，acCTable域就包含系统设定控制部件色彩表的间接指针。

当绘制一个控制部件时，标准的控制部件定义函数在链接的辅助控制记录列表中搜索其acOwner域指向被绘制的控制部件的辅助控制记录。如果标准的控制部件定义函数找到控制部件的辅助控制记录，它们就使用在acCTable域中指定的控制部件色彩表。如果标准的控制部件定义函数没有找到控制部件的辅助控制记录，它们就使用系统设定的系统色彩。

弹出式菜单专用数据记录

通过间接引用弹出式菜单的控制记录的contrlData域，您可获得与弹出式菜单相关的菜单的菜单间接指针和菜单标识符。控制记录的contrlData域是专用信息块的间接指针。对于弹出式菜单控制部件，这个域是弹出式专用数据记录的间接指针，弹出式专用数据记录是一个popupPrivateData类型的数据结构。

```
TYPE    popupPrivateData =
        RECORD
            mHandle: MenuHandle;    {handle to menu record}
            mID:     Integer;       {menu ID}
            mPrivate:               ARRAY[0..0] OF SignedByte;
                                    {reserved}
        END;
```

地域说明

mHandle 包含菜单的间接指针。
m1D 菜单的菜单标识符。
mPrivate 保留。

您可使用标准的弹出式控制部件定义函数来管理弹式或菜单。若想了解关于创建弹出式菜单的信息，请参阅5 - 25页的“创建弹出式菜单”。若想了解相关的补充信息，请参阅本书“菜单管理器”一章。

控制部件色彩表记录

通过使用SetControlColor过程来创建控制部件色彩表记录（在5 - 101页中有介绍），您的应用程序可以使用系统设定以外的其它彩色绘制控制部件。（另一方面，通过创建与控制资源具有相同的资源标识符的控制部件色彩表资源，您可以将非标准的彩色用于您在控制资源中定义的控制部件。在5 - 121页有关于创建控制部件色彩表资源的介绍。）要意识到使用非标准彩色的控制部件最初可能使用户感到迷惑。

控制部件色彩表记录是一个CtlCTab类型的数据结构；它是这样定义的:

```
TYPE CtlCTab =
RECORD
    ccSeed:  LongInt;       {reserved; set to 0}
    ccRider: Integer;       {reserved; set to 0}
    ctSize:  Integer;       {number of ColorSpec records in next }
                            { field; 3 for standard controls}
    ctTable: ARRAY[0..3] OF ColorSpec;
END;
```

地域说明

ccSeed 保留在控制部件色彩表中；设定为0。

ccRider 保留在控制部件色彩表中；设定为0。

ctSize 在下一个域中 ColorSpec 记录的数目。对于用标准定义过程绘制的控制部件这个域总是 3。因为一个标准控制部件有3部分：框架、控制体和滚动杆的滚动块；或者框架、控制体和其它控制部件的文字说明。如果您要为额外的部分提供ColorSpec记录，您就必须定义您自己的控制部件。5－109页的“定义您自己的控制部件定义函数”中有相关的介绍。

ctTable 是一个ColorSpec记录阵列。每个ColorSpec记录说明一个不同控制部件的彩色。这里介绍一个ColorSpec记录是如何定义的:

```
TYPE ColorSpec =
RECORD
     partIdentifier:    Integer;      {control part}
     partRGB:           RGBColor;     {color of part}
END;
```

ColorSpec记录的partIdentifier域中存储一个整数，它使得RGBColor记录与一个特定的控制部件组成部分相关联。

三个ColorSpec记录用来说明按钮、复选框和选项按钮部分。下面是用来说明这些控制部件的三个 ColorSpec 记录的 partIdentifier 域中使用的常数:

```
{for buttons, checkboxes, and radio buttons}
CONST     cFrameColor = 0;          {frame color}
          cBodyColor = 1;           {fill color for body of }
                                    { control}
          cTextColor = 2;           {text color}
```

当醒目显示时，按钮改变它们的外观和文字的色彩；复选框和选项按钮只改变它们的外观而不改变色彩。但都通过使它们的文字变暗来表示处于无效状态，而不对其色彩作任何变化。

有一些ColorSpec记录被用来说明滚动杆的那五个控制部分。下面是用来说明滚动杆色彩的ColorSpec记录的partIdentifier域中使用的常数:

常数

```
cFrameColor        = 0; {Used to produce foreground color for scroll arrows }
                        { & gray area}
cBodyColor         = 1; {Used to produce colors in the scroll box}
cArrowsColorLight  = 5; {Used to produce colors in arrows & scroll bar }
                        { background color}
cArrowsColorDark   = 6; {Used to produce colors in arrows & scroll bar }
                        { background color}
cThumbLight        = 7; {Used to produce colors in scroll box}
cThumbDark         = 8; {Used to produce colors in scroll box}
cHiliteLight       = 9; {Use same value as wHiliteColorLight in 'wctb'}
cHiliteDark        = 10;{Use same value as wHiliteColorDark in 'wctb'}
cTitleBarLight     = 11;{Use same value as wTitleBarLight in 'wctb'}
cTitleBarDark      = 12;{Use same value as wTitleBarDark in 'wctb'}
cTingeLight        = 13;{Use same value as wTingeLight in 'wctb'}
cTingeDark         = 14;{Use same value as wTingeDark in 'wctb'}
```

当醒目显示时，滚动箭头填以前景彩色。无效的滚动杆不显示滚动块，并且用背景色显示它的灰度区域，同时不在其中显示任何图案。

一个控制部件的ColorSpec记录可以任何次序出现。如果您包括一个没有找到的部分标识符，控制管理器就使用第一个具有可标识部分的ColorSpec记录。如果您不指定部分标识符，控制管理器就对这个部分使用系统设定的色彩。

ColorSpec记录的partRGB域指定一个RGBcolor记录，这个记录又指定部分彩色的红、绿、蓝色的值。使用 3 个 16 位无符号整数给出 3 个可叠加基色的强度值。这里说明RGBColor记录是怎样定义的:

```
TYPE RGBColor =
RECORD
     red:     Integer; {red value for control part}
     green:   Integer; {green value for control part}
     blue:    Integer; {blue value for control part}
     END;
```

当您创建一个控制部件色彩表记录时，如果另一个控制部件还在使用它，您的应用程序就不应该对它解除内存分配。

当绘制一个控制部件时，标准的控制部件定义函数在链接的辅助控制记录列表中搜索其acOwner域指向这个控制部件的记录。如果标准控制部件定义函数找到这样一个记录，它就使用这个记录指定的色彩表；否则它使用系统设定的色彩。每一个使用系统设定以外的彩色的控制部件有它自己的辅助控制记录，尽管这个控制部件使用与另一个控制部件相同的控制部件色彩表记录，两个或多个辅助记录可以共享同样的控制部件色彩表记录。（在5 – 76页有关于辅助控制记录的介绍。）

如果您创建一个控制部件定义函数（在5 – 109页的“定义您自己的控制部件定义函数”中有这方面的解释。），除 1 至 127 是保留给系统定义外。您可以使用任何大小的色彩表并随心所欲地定义它们的内容。任何这样的非标准控制部件定义函数应该通过给它创建的每个控制部件指定一个显式辅助记录来绕过系统设定机制。

控制管理器例程

本节介绍控制管理器例程，它们用于创建控制部件、绘制控制部件、跟踪控制部件中的鼠标事件、改变控制部件显示、确定控制部件设定值和取消控制部件。

一些控制管理器例程可以使用多于一个的例程名称来存取，这取决于您的开发环境支持的界面文件。例如，SetCtlValue与SetControlValue是等效的。表5 – 1提供了例程以前的名称和它的新等效名称之间的映像。

表 5 – 1 控制管理器例程的新名称与旧名称之间的映射

新名称	旧名称
GetAuxiliaryControlRecord	GetAuxCtl
GetControlAction	GetCtlAction
GetControlMaximum	GetCtlMax
GetControlMinimum	GetCtlMin
GetControlReference	GetCRefCon
GetControlTitle	GetCTitle
GetControlValue	GetCtlValue
GetControlVariant	GetCVariant
SetControlAction	SetCtlAction
SetControlColor	SetCtlColor
SetControlMaximum	SetCtlMax
SetControlMinimum	SetCtlMin
SetControlReference	SetCRefCon
SetControlTitle	SetCTitle
UpdateControls	UpdtControl

创建控制部件

若要创建一个控制部件，您通常应使用 GetNewControl 函数，这个函数采用源自控制资源的关于控制部件的信息。像菜单资源一样，控制资源将说明性的信息从您的应用程序代码中分离出来，使得您的应用程序修改和转换起来更容易。您也可使用 NewControl 函数（您在参数中为它传递说明性信息）来创建控制部件。

GetNewControl和NewControl均返回最新创建的控制部件的控制记录间接指针。此后，您的应用程序一般通过这个间接指针访问控制部件，因为大多数其它控制管理器例程把控制间接指针作为它们的第一个参数。为您创建滚动杆和弹出式菜单时，您应该在您自己的一个数据结构中存储它们的间接指针以便将来参考。

当您使用对话管理器在警告栏和对话框中实现按钮、选项按钮、复选框和弹出式菜单时，对话管理器自动地使用控制管理器为您创建这些控制部件。如果您在警告或对话框中实现其它控制部件，或不管什么时候在您的应用程序的窗口中实现控制部件（如滚动杆），您都必须使用 GetNewControl 或 NewControl 来创建这些控制部件。

GetNewControl

若要依据控制资源（'CNTL'）中的说明创建一个控制部件，请调用 GetNewControl 函数。

```
FUNCTION GetNewControl(controlID: Integer; Owner: WindowPtr):
:ControlHandle;
```

controlID	控制资源的资源标识符。
Owner	您想在其中附加控制部件的窗口的指针。

说明

GetNewControl 函数依据指定的控制资源中的信息创建一个控制记录，把控制记录添加到指定窗口的控制列表中，并把控制部件的间接指针作为它的函数结果返回。当您在大多数其它控制管理器例程中访问控制部件时您使用这个间接指针。在拷贝了一个控制资源后，GetNewControl 释放原先的控制资源在返回前占据的存储器。

如果您提供一个与控制资源具有相同的资源标识符的控制部件色彩表资源，GetNewControl 就创建一个辅助控制记录，它使用您在您的控制部件色彩表资源中指定的彩色。如果您没有提供控制部件色彩表，如果电脑正运行于 32 位模式，Get-NewControl 就使用系统设定的控制部件色彩表创建一个辅助控制记录。

控制资源指定控制部件的矩形、它的初始设定状态、它的可视性状态、它的最大和最小设定、它的控制部件定义标识符、一个参考值和它的标题（如果有的话）。在您使用 GetNewControl 创建控制部件后，您可分别地使用 SetControlValue、SetContro-lMaximum、SetControlMinimum、SetControlReference 和 SetControlTitle 过程来更改当前设定状态、最大设定状态、最小设定状态、参考值和标题。您可使用 MoveControl 和 SizeControl 过程来更改控制部件的矩形。您可使用 GetContro-lValue、GetControlMaximum、GetControlMinimum、GetControlReference 和 GetControlTitle 函数来确定控制值。

如果控制资源指定控制部件应该可视，控制管理器就绘制控制部件。如果控制资源指定控制部件最初应该不可视，您就可使用 ShowControl 过程使控制部件可视。

如果 GetNewControl 不能从资源文件中读取控制资源，GetNewControl 就返回 NIL。

亦请参阅

请参阅 5 - 17 页的程序清单 5 - 1 和 5 - 24 页的程序清单 5 - 5，关于如何使用 GetNewControl 来分别创建按钮和滚动杆。若想了解关于窗口的控制列表的信息，请参阅本书“窗口管理器”一章。

NewControl

若要创建一个控制部件，您可使用 NewControl 函数，它在它的参数中接受说明控制部件的信息。一般说来，您应该使用 GetNewControl 函数来创建一个控制部件。GetNewControl函数采用源自一个控制资源关于控制部件的信息，这样做的结果就是对您的应用程序进行修改或转换成其它语言就会更容易。

```
FUNCTION NewControl (theWindow: WindowPtr; boundsRect: Rect;
                     title: Str255; visible: Boolean;
                     value: Integer; min: Integer; max: Integer;
                     procID: Integer; refCon: LongInt)
                     : ControlHandle;
```

theWindow 您想要在其中附加控制部件的窗口的指针。属于控制部件的所有坐标均在这个窗口的局部坐标系统中解释。

boundsRect 在给定窗口的局部坐标中指定的矩形，这个矩形环绕控制部件，因而确定控制部件的大小和位置。

title 需要标题的控制部件（如按钮、复选框、选项按钮和弹出式菜单）的标题字符串。对于那些不使用标题的控制部件，则传递一个空字符串。

visible 控制部件的可视／不可视状态。如果您在这个参数中传递 TRUE，NewControl 就马上绘制控制部件，并不使用您的窗口的标准更新机制。如果您传递 FALSE，您必须随后使用 ShowControl 过程来显示这个控制部件。

value 控制部件的初始设定状态，对于不保留设定状态的控制部件（如按钮），在这个参数中传递 0。对于保留一个开或关设定状态的控制部件（如复选框和选项按钮），在关闭的控制部件的这个参数中传递 1。对于那些可采用一定范围设定状态的控制部件（如滚动杆和滑块），指定在这个范围内合适的任何值。

min 控制部件的最小设定状态。对于不保留设定状态的控制部件（如按钮），在这个参数中传递 0。对于保留一个开或关设定状态的控制部件（如复选框和选项按钮），使用 0（表示“关”）作为最小值。对于那些可采用一定范围设定状态的控制部件（如滚动杆和滑块），指定任意合适的最小值。

max 控制部件的最大设定状态。对于不保留设定状态的控制部件（如按钮），该参数中传递 1。对于保留一个开或关设定状态的控制部件（如复选框和选项按钮），使用 1（表示“开”）作为最大值。对于那些可采用一定范围的设定状态的控制部件（如滚动杆和滑块），指定任意合适的最大值。当您使一个滚动杆的最大设定状态等于其最小设定状态时，控制部件定义函数将自动地使滚动杆变成无效的；当您使最大设定状态超出最小设定状态时，控制部件定义函数将使滚动杆重新变成有效的。

procID 控制部件定义标识符，指向这种类型控制部件的控制部件定义函数。如果控制部件定义函数不在存储器中就把它读进存储器。控制部件定义标识符及其用于标准控制部件的常数列表如下。（您也可定义您自己的控制部件定义函数并指定它的 procID 参数）

常数

```
pushButProc    = 0;      {button}
checkBoxProc   = 1;      {checkbox}
radioButProc   = 2;      {radio button}
useWFont       = 8;      {add to above to display }
                         { title in window font}
scrollBarProc = 16;      {scroll bar}
popupMenuProc = 1008;    {pop-up menu}
popupFixedWidth= $0001;{add to popupMenuProc to }
                         { use a fixed-width ctrl}
```

```
        popupUseAddResMenu= $0004; {add to popupMenuProc to }
                                   { specify a value of }
                                   { type ResType in the }
                                   { contrlRfCon field of }
                                   { the control record; }
                                   { Menu Manager adds }
                                   { resources of this type }
                                   { to the menu}
            popupUseWFont = $0008; {add to popupMenuProc to }
                                   { display in window font}
```

refCon　控制部件的参考值，它仅由您的应用程序设定和使用。

NewControl 函数依据您在其参数中指定的信息创建控制记录，把控制记录添加至指定窗口的控制列表中，并返回控制部件的间接指针作为它的函数结果。当您在大多数其它的控制管理器例程中访问控制部件时都使用到这个间接指针。

如果电脑正运行于 32 位模式，NewControl 函数就创建一个使用系统设定控制部件色彩表的辅助控制记录。

如果您需要在控制部件中使用除系统设定色彩以外的彩色，就创建一个控制部件色彩表记录并使用 SetControlColor 过程。

当在 boundsRect 参数中指定矩形时，记住以下准则:

- 按钮的绘制要与矩形十分吻合。为了适应系统字体中最高的字符，至少允许矩形的顶部和底部坐标间有 20 个像素之差。
- 对于复选框和选项按钮来说，顶部和底部坐标之间至少应有 16 个像素之差。
- 按照惯例，滚动杆为 16 个像素宽，所以在左和右（或顶部与底部）坐标之间应该有 16 个像素之差。（如果没有，滚动杆就按比例缩至与矩形吻合。）一个标准的滚动杆应该至少有 48 个像素长，以便能容纳滚动箭头和滚动块。

控制管理器采用系统字体显示控制部件标题。当在 title 参数中指定控制部件的标题时，一定要保证标题与控制部件的矩形相吻合；否则，NewControl 就会截短标题。例如，NewControl 在右边以罗马语系截短复选框和选项按钮的标题，对于按钮标题，它则令标题居中，因而从两端截去字符。

说明

控制管理器允许在按钮、复选框和选项按钮的标题中有多行文字。当指定多行标题时，用 ASC Ⅱ字符代码 $ OD（回车）分隔各行。如果控制部件是一个按钮，每行就按水平方向居中。(每行的高度等于上一行至下一行的矩离加上所用的行间距。一定要使矩形的总高度大于行数与每行高度之乘积。)如果控制部件是一个复选框或选项按钮，文字就按适合于用户的当前语系系统的方式对齐，并且复选框或按钮在其矩形内是按垂直方向进行居中定位的。

在您使用 NewControl 创建控制部件后，您可以分别使用 SetControlValue、SetControlMaximum、SetControlMinimum、SetControlReference 和 SetControlTitle 过程来改变当前设定状态、最大设定状态、最小设定状态、参考值和标题。您可使用 SetControl 和 SizeControl 过程来改变控制部件的矩形。您可以使用 GetControlValue、GetControlMexium、GetControlMinimum、GetControlReference 和 GetControlTitle 函数来确定控制值。

特殊考虑

按钮、复选框、选项按钮或弹出式菜单的标题一般以系统字体出现，系统字体在 Roman 语系系统中是 12 点的 Chicago。不要使用更小的字体；一些语系系统，如 KanjiTalk，要求 12 点的字体。您通常应该在您的控制部件中使用系统字体；这样做将简化本地化工作。但是，如果您一定需要用当前与窗口的图形端口相关联的字体显示控制标题，您就可把 popupUseWFont 常数添加至弹出式菜单控制部件定义标识符，或把 useWFont 常数添加至其它标准的控制部件定义标识符。

亦请参阅

若想了解关于窗口的控制列表的信息，请参阅本书“窗口管理器”一章。5 - 109 页“定义您自己的控制部件定义函数”中有关于其它控制部件的控制部件定义标识符的讨论。

绘制控制部件

如果您（在控制资源功或NewControl的参数中）指定一个控制部件一开始就可视的，当您调用 GetNewControl 或者 NewControl 函数时控制管理器就在控制部件的窗口内绘制控制部件。这两种情形中，控制管理器均立即绘制控制部件，而不使用您的窗口的标准更新机制。如果您指定一个控制部件不可视，当您想要绘制控制部件时您就可使用 ShowControl 过程。

注意，即使是一个可视控制部件，它也可能被叠加窗口或其它对象完全或部分地遮住。

当您的应用程序接收到包含控制部件的窗口的一个更新事件时，使用 UpdateControls 在更新的窗口中重新绘制必要的控制部件。注意，对话管理器在警告栏和对话框中自动地绘制和更新控制部件。

ShowControl

若要绘制一个当前不可视控制部件，您可使用 ShowControl 过程。

```
PROCEDURE ShowControl ( the Control: ControlHandle )£ª
```

theControl　　您想使其变成可视的控制部件的间接指针。

说明

如果指定的控制部件不可视，ShowControl过程就使它变成可视并立即在其窗口内绘制控制部件，而不使用您的窗口的标准更新机制。如果控制部件已经是可视的，ShowControl 就不起任何作用。

您可用几种方法使一个控制部件不可视:
- 您可在它的控制资源中指定它不可视
- 您可在 NewControl 函数的参数中指定它不可视
- 您可使用 HideControl 过程把一个可视控制部件变成不可视
- 您可直接更改控制部件的控制记录的 contrlVis 域

特别考虑

ShowControl过程在控制部件的窗口内绘制控制部件，但控制部件仍可能被重叠窗口或其它对象全部或部分地遮住。

亦请参阅

5 – 39页程序清单5 – 14介绍了在移动和调整滚动杆大小后，使用ShowControl来重新显示滚动杆。

UpdateControls

若要更新窗口中的控制部件，您可使用 UpdateControls 过程。UpdateControls 过程也就是 UpdateControl 过程。

```
PROCEDURE UpdateControls (theWindow: WindowPtr; UpdatRgn:
RgnHandle);
```

theWindow 包含要更新的控制部件的窗口的指针。

UpdateRgn 指定窗口内的更新区域。

说明

UpdateControls 过程绘制那些在指定的更新区域中的控制部件。这个过程比 DrawControls 过程要快。DrawControls 过程绘制窗口内的所有控制部件。与此对照，UpdateControls 仅绘制那些在更新区域中的控制部件。

如果一个窗口包含控制部件，您的应用程序在接收到关于这个窗口的更新事件时就应调用 UpdateControls。窗口管理器例程，如 SelectWindow、ShowWindow 和 BringToFront，并不自动调用 DrawControls 来显示窗口的控制部件。它们只是把相应的区域添加到窗口的更新区域，然后生成一个更新事件。

在响应一个更新事件时，您一般在使用窗口管理过程 BeginUpdate 之后，使用窗口管理器过程 EndUpdate 之前，调用 UpdatControls。您应该把 UpdateRgn 参数设定成窗口端口的 VisRgn 域所指定的端口可视区域。

特别考虑

如果您的应用程序把控制部件的部分绘制到其矩形之外，UpdateControls 不会重新绘制这个控制部件。

对话管理器处理在警告栏和对话框内的控制部件的更新事件。

亦请参阅

5 - 30 页的程序清单 5 - 8 说明了如何使用 UpdateControls。本书的“窗口管理器”一章介绍了 BeginUpdate 和 EndUpdate 过程。若想了解关于在警告栏和对话框中包括控制部件的更多信息，请参阅本书“对话管理器”一章。

DrawControls

尽管您通常使用 UpdateControls 过程来更新窗口中的控制部件，您也可以使用 DrawControls 过程。

```
PROCEDURE DrawControls ( theWindow: WindowPtr );
```

theWindow　　您想要显示其控制部件的窗口的指针。

说明

DrawControls 过程绘制在指定窗口内当前可视的所有控制部件。按照与创建这些控制部件时相反的次序绘制这些控制部件；因而，对于那些叠加的控制部件，最先创建的控制部件出现在窗口的最前面。

因为 UpdateControls 过程仅重新绘制那些需要更新的控制部件，所以在接收到包含控制部件的窗口的更新事件时您的应用程序一般应该使用 UpdateControls 过程而不是 DrawControls 过程。

在调用窗口管理器过程 BeginUpdate 之后、调用 EndUpdate 之前，您应该调用 DrawControls 和 UpdateControls 这两个过程其中之一。

特殊考虑

对话管理器自动地绘制和更新在警告栏和对话框中的控制部件。

窗口管理器例程，如SelectWindow、ShowWindow和BringToFront，并不自动更新窗口的控制部件。它们只把相应的区域添加至窗口的更新区域，并生成一个更新事件。

亦请参阅

若想了解关于在警告栏和对话框中包括控制部件的更多的信息请参阅本书“对话管理器”一章。若想了解关于窗口管理器例程更多的信息，请参阅本书“窗口管理器”一章。

Draw1Control

尽管您一般应该使用UpdateControls过程来更新控制部件，您也可使用Draw-1Control 过程来更新某个控制部件。

```
PROCEDURE Draw1Control (theControl: ControlHandle);
```

theControl　您想要绘制的控制部件的间接指针。

说明

如果指定的控制部件在其窗口内是可视的，Draw1Control 过程就绘制这个控制部件。UpdateControls 过程自动地调用 Draw1Control。

处理控制部件中的鼠标事件

当用户按下鼠标按钮时，您的应用程序接收到一个鼠标按下事件。使用窗口管理器函数 FindWindow 确定光标在哪个窗口中。如果鼠标按下事件发生在您的应用程序的当前活动窗口的内容区域，使用FindControl函数来确定光标是否在一个有效的控制部件中，如果在，是在哪个控制部件中。若要跟踪和响应那个控制部件中的移动鼠标、并随后确定释放鼠标事件发生在控制部件中的哪个部分，使用 TrackControl 函数。

FindControl

若要确定一个鼠标按下事件是否发生在一个控制部件中，以及在那个控制部件的哪个部分中（如果是发生在控制部件中的话），请使用 FindControl 函数。

```
FUNCTION FindControl(thePoint:Point;theWindow:WindowPtr;
                    VAR theControl:ControlHandle):Integer;
```

thePoint 在窗口局部坐标中指定的鼠标按下事件发生的点。
theWindow 鼠标按下事件发生所在的窗口的指针。
theControl 鼠标按下事件发生所在的控制部件的间接指针。

说明

当光标在一个可视的、有效的控制部件中时，用户按下鼠标按钮，FindControl 返回一个标识控制部件某部分的部分代码作为它的函数结果。函数还在参数 theControl 中返回控制部件的间接指针。FindControl 返回的部分代码以及您可用来代表它们的常数，程序清单如下：

```
常数 inButton = 10;          {button}
     inCheckBox    = 11;     {checkbox or radio button}
     inUpButton    = 20;     {up arrow for a vertical scroll }
                             {bar, left arrow for a horizontal}
                             { scroll bar}
     inDownButton = 21;      {down arrow for a vertical scroll}
                             { bar, right arrow for a }
                             { horizontal scroll bar}
     inPageUp = 22;          {gray area above scroll box for a}
                             { vertical scroll bar, gray area }
                             { to left of scroll box for a }
                             { horizontal scroll bar}
     inPageDown    = 23;     {gray area below scroll box for a}
                             { vertical scroll bar, gray area }
                             { to right of scroll box for a }
                             { horizontal scroll bar}
     inThumb  = 129;         {scroll box}
```

弹出式控制部件定义函数没有定义弹出式菜单的部分代码。但您的应用程序在创建弹出式菜单时应该存储它们的间接指针。在响应弹出式菜单的用户选择之前，您的应用程序应该对照 FindControl 返回的间接指针测试您存储的间接指针。

如果鼠标按下事件发生在一个不可视或无效的控制部件中，或者如果它发生在控制部件之外，FindControl 把 theControl 设定成 NIL 并返回 0 作为它的函数结果。

当一个鼠标按下事件发生时，在使用窗口管理器函数 FindWindow 确定鼠标按下事件发生在包含控制部件的窗口的内容区域之后，您的应用程序应该调用 FindControl。

在调用 FindControl 之前，使用 GlobalToLocal 过程把存储在事件记录的 Where 域（它说明鼠标按下事件的位置）中的点转换成窗口的局部坐标。然后，当使用 FindControl 时，在参数 thePoint 中传递这个坐标点。

在参数 theWindow 中，传递 FindWindow 函数返回的窗口指针。

在使用 FindControl 确定一个鼠标按下事件发生在一个控制部件中之后，您一般使用 TrackControl 函数，这个函数将自动地跟踪光标移动并作出响应直到用户释放鼠标按钮为止。

特别考虑

对于在警告栏和对话框的控制部件内的鼠标按下事件，对话管理器将自动地调用 FindControl 和 TrackControl。

如果窗口不可视，或如果窗口不包含给定的点，FindControl 函数就在参数 theControl 中也返回 NIL，并返回 0 作为它的函数结果。（然而，FindWindow 将不返回不可视窗口的窗口指针，也不返回不包含鼠标按下事件发生所在点的窗口的指针。只要您在调用 FindControl 之前调用 FindWindow，这种情况就不会出现。）

亦请参阅

5 – 33 页的程序清单 5 – 10 说明了如何使用 FindControl 来探测在弹出式菜单和按钮中的鼠标按下事件。5 – 53 页的程序清单 5 – 18 说明了如何用它来探测在滚动杆中的鼠标按下事件。

在本书的“窗口管理器”一章中介绍了 FindWindow 函数。在《Inside Macintosh 绘图》*(Inside Macintosh: imaging)* 中介绍了 GlobalToLocal 过程。

在本书的“事件管理器”一章中有关于事件记录的介绍。若想了解关于在警告栏和对话框中包括控制部件的更多的信息，请参阅本书“对话管理器”一章。

TrackControl

若要在一个控制部件中跟踪和响应光标移动，确定释放鼠标事件发生在哪个控制部分，可以使用 TrackControl 函数。

```
FUNCTIONTrackControl (the ControlControlHandle; thePoint:
                        Point; actionProc:ProcPtr) Integer;
```

theControl 鼠标按下事件在其中发生的控制部件的间接指针。

thePoint 在窗口局部坐标中指定的鼠标按下事件发生所在的点。

actionProc 动作过程。一般说来，对于按钮、复选框、选项按钮和滚动杆的滚动块，您的应用程序应该把这个参数设定成 NIL；对于弹出式菜单，把这个参数设定为 Pointer (-1)；对于滚动箭头和滚动杆的灰度区域，以及当用户按住鼠标按钮时要求您定义采取额外动作的任何其它的控制部件，把这个参数设定成动作过程的指针。

说明

TrackControl 函数在控制部件中跟踪用户的光标移动，并提供直观的反馈直到用户释放鼠标按钮为止。由TrackControl给出的直观反馈取决于鼠标按下事件发生所在的控制部分。例如，当醒目显示有效时，TrackControl 醒目显示这个控制部分（并且当用户释放鼠标按钮时取消其醒目显示）。当光标在一个指示符（如滚动杆的滚动块）中用户按下鼠标按钮并移动鼠标时，TrackControl作出响应拖拽这个指示符的虚线轮廓。

如果光标处在控制部件中时用户释放鼠标按钮，TrackControl 函数就返回控制部件的部件代码作为它的函数结果。如果光标在控制部件分之外时用户释放鼠标按钮，TrackControl 函数就返回 0。TrackControl 函数返回的控制部件的值与 FindControl 函数所返回的一样（均由常数 inButton、inCheckBox、inUpButton、inDownButton、inPageUp、inPageDown 和 inThumb 表示），在 5 － 89 页中有相关的介绍。

当 TrackControl 返回一个非 0 的值作为它的函数结果时，您的应用程序应该相应地响应在那个控制部件中的释放鼠标事件。当 TrackControl 返回 0 时，您的应用程序应该不予理睬。

如果光标在一个指示符（如一个滚动块）中时用户释放按钮，TrackControl 就调用控制部件的控制部件定义函数重新定位指示符。例如，滚动杆的控制部件定义函数在响应用户拖拽滚动块时，重新绘制滚动块，根据滚动块新的相对位置计算控制部件的当前设定状态，并在控制记录中存储当前设定状态。如果最小和最大设定状态是0和10，滚动块处在滚动杆的中间，那么 5 就被作为当前设定状态存储起来。对于滚动杆来说，您的应用程序响应时就必须滚动到文稿中相应的相对位置。

通常，您在调用 FindControl 函数后调用 TrackControl。在 TrackControl 的参数 theControl 中，传递由 FindControl 函数返回的控制间接指针，并在参数 thePoint 中提供与您传递给 FindControl 相同的点（在窗口局部坐标中的点）。

当光标在一个标准控制部件中用户按住鼠标按钮时，TrackControl 就根据您在参数 actionProc 中传递的值执行下列动作。（对于其它控制部件，您在这个参数中传递什么取决于您是怎样定义控制部件的。）

- 如果您在 actionProc 参数中传递 NIL，TrackControl 就不使用任何动作过程，因而除了醒目显示控制部件或拖拽指示符外不执行任何额外动作。这适用于按钮、复选框、选项按钮和滚动杆的滚动块
- 如果您在 actionProc 参数中传递一个动作过程的指针，您就必须提供这个过程，并且只要用户按住鼠标按钮它还必须定义一些您的应用程序重复执行的动作。这适用于滚动箭头和滚动杆的灰度区域
- 如果您在 actionProc 参数中传递 Pointe (-1)，TrackControl 就在控制记录的 ControlAction 域中寻找控制部件的动作过程指针。这适用于您在弹出式菜单中跟踪光标。（您通过调使用 GetControlAction 函数来确定这个域的值，用 SetC-ontrolAction 过程来更改这个值。）如果控制记录的 ControlAction 域包含一个过程指针，TrackControl 就使用它所指向的动作过程。如果控制记录的域也包含值 Pointer (-1)，TrackControl 就调用控制部件的控制部件定义函数来执行必要的动作。如果您为一个自定义控制部件定义您自己的控制部件定义函数，您就可能希望这样做。如果控制记录的域包含值 NIL，TrackControl 就不执行任何动作

特别考虑

当您需要处理在警告栏和对话框中的事件时，对话管理器例程自动地调用 FindControl 和 TrackControl。

汇编语言信息

TrackControl 函数调用窗口管理器函数 DragGrayRgn，这样您可以使用全局变量 DragHook 和 DragPattern。

亦请参阅

若想了解关于在 actionProc 参数中指定的动作过程的信息，请参阅 5 — 115 页的“定义您自己的动作过程。”若想了解关于创建控制部件定义函数的信息，请参阅 5 — 109 页的“定义您自己的控制部件定义函数。”

5 — 36 页的程序清单 5 — 11、5 — 37 页的程序清单 5 — 12、5 — 38 页的程序清单 5 — 13 和 5 — 53 页的程序清单 5 — 18，分别说明了如何使用 TrackControl 来响应在按钮、弹出式菜单、复选框和滚动杆中的鼠标按下事件。

若想了解关于在警告栏和对话框中包括控制部件的更多信息，请参阅本书“对话管理器”一章。

TestControl

FindControl 和 TrackControl 都调用 TestControl 函数，正常情况下，您不需要自己调用它。但是，每当您需要确定发生鼠标按下事件的控制部件，您就可以使用 TestControl 函数。

```
FUNCTION TestControl (theControl: ControlHandle; thePt: Point)
                        : Integer;
```

theControl　　控制部件的一个间接指针，鼠标按下事件在该控制部件中发生。

thePt　　发生鼠标按下事件的位置信息，以窗口的局部坐标表示。

说明

当参数 theControl 指定的控制部件可见并有效时，TestControl 测试参数 thePt 指定的点在控制部件的哪一部分。TestControl 返回该点所在的控制部分的部分代码作为函数结果，如果该点在控制部件之外，它返回 0。

更改控制部件设定值和显示

为了响应用户的动作，您常常需要更改控制部件的设定值、醒目显示状态、大小和位置。每当您的应用程序调用 TrackControl 函数时，控制管理器就在用户按下和释放鼠标按钮时，自动操纵控制部件的适当的显示。例如，TrackControl 调用 HiliteControl 过程醒目显示按钮；对于滚动杆，TrackControl 调用 DragControl 过程在滚动杆中移动滚动块的轮廓线，调用 SetControlValue 过程更改滚动杆的当前设定值并在新的位置重新绘制滚动块。(注意，对话管理器自动调用 TrackControl 处理警告栏和对话框中的控制部件。请参阅本书的“对话管理器”一章中有关的更详细信息。)

如果用户释放鼠标按钮的同时，光标在一个控制部件中，您的应用程序常常需要更改该控制部件的设定值。例如，当用户单击一个复选框时，您的应用程序必须把它的设定值更改成启用或停用，控制管理器自动地绘制或取消复选框中的选择标记“X”。

还有许多必须更改控制部件的设定值和显示的例子。例如，当用户更改包含滚动杆的窗口的大小时，您需要相应地调整滚动杆的大小并移动它。

对于值由用户设定的控制部件，您可以使用SetControlValue过程更改控制部件的设定值，并相应地重新绘制控制部件。当您需要更改滚动杆或一个拨号的最大设定值时，您可以使用SetControlMaximum过程；如果您需要更改最小设定值，可以使用SetControlMinimum过程。如果您需要更改控制部件标题，可以使用SetControlTitle过程。您可以使用HideControl过程使控制部件不可见。当您需要使一个控制部件无效时(例如当它的窗口不在最前面时)或者以其它方式更改控制部件的醒目显示时，您可以使用HiliteControl过程。

若要移动一个滚动杆，您应使用MoveControl和SizeControl过程。

您也可以使用SetControlColor过程把控制部件的系统设定色彩更改成您自己选择的彩色，但是我们并不推荐您这样做。

为了在用户按住鼠标按钮时，请求执行连续的动作，您可以在TrackControl的一个参数中指定一个动作过程（请参阅 5-115 页的“定义您自己的动作过程”一节）。在一定的条件下，您可以使用SetControlAction过程更改控制部件的动作过程，但是您应该尽可能不这样作。

SetControlValue

若要更改控制部件的设定值，并相应地重新绘制它，您可以使用SetControlValue过程。SetControlValue过程和SetCtlValue过程是一样的。

```
PROCEDURE SetControlValue (theControl: ControlHandle; theValue:
Integer);
```

theControl　　控制部件的一个间接指针，您希望更改该

控制部件的当前设定值。

theValue　　控制部件的新设定值。

亦请参阅

SetControlValue过程把控制记录的contrlValue域更改为指定的值，并按照新的设定值，重新绘制该控制部件。对于复选框和选项按钮，值 1 为在控制部件中填上适当的标记，值 0 删除该标记。对于滚动杆，SetControlValue在适当的地方重新绘制滚动块。

如果指定值小于控制部件的最小设定值，SetControlValue就设定控制部件的最小值。如果指定值大于最大设定值，SetControlValue就设定控制部件的最大值。

当您创建一个控制部件时，您在控制资源或NewControl函数的value参数中指定一个初始设定值。在响应用户点按控制部件而更改它的当前设定值之前，应该首先确定这个值，您可以使用GetControlValue函数获得控制部件的当前设定值。

说明

5-38 页的程序清单 5-13 解释了使用 SetControlValue 更改复选框设定值的方法。5-41 页的程序清单 5-16 和 5-61 页的程序清单 5-20 解释了使用 SetControlValue 改变滚动杆设定值的方法。

SetControlMinimum

若要更改控制部件的最小设定值，并相应地重新绘制它的指示符或滚动块，您可以使用 SetControlMinimum 过程。SetControlMinimum 过程和 SetCtlMin 过程是一样的。

```
PROCEDURE SetControlMinimum (theControl: ControlHandle; minValue:
Integer);
```

theControl　控制部件的一个间接指针，您希望更改该控制部件的最小设定值。

minValue　新的最小设定值。

说明

SetControlMinimum 过程把控制记录的 contrlMin 域更改为您在 minValue 中指定的设定值，并按照它的新范围重新绘制它的指示符或滚动块。

当您创建控制部件时，您在控制资源中或在 NewControl 函数的 min 参数中指定一个初始最小设定值。为了确定控制部件的当前最小设定值，应使用 GetControlMinimum 函数。

SetControlMaximum

若要更改控制部件的最大设定值，并相应地重新绘制它的指示符或滚动块，您可以使用 SetControlMaximum 过程。SetControlMaximum 过程和 SetCtlMax 过程是一样的。

```
PROCEDURE SetControlMaximum (theControl: ControlHandle; maxValue:
Integer);
```

theControl　控制部件的一个间接指针，您希望更改该控制部件的最大设定值。

maxValue　新的最大设定值。

说明

SetControlMaximum 过程把控制记录的 contrlMax 域更改为您在 maxValue 中指定的设定值，并按照它的新范围重新绘制它的指示符或滚动块。

动杆的最大设定值和最小设定值设定为相等时，控制部件定义函数使滚动杆无效. 当您令最大设定值超过最小设定值时，控制部件定义函数使滚动杆重新有效。

亦请参阅

5-41 页的程序清单 5-16 解释了使用SetControlMaximum指定滚动杆的最大设定值的方法。

SetControlTitle

若要更改控制部件的标题，并相应地重新绘制控制部件，应使用 SetControlTitle 过程。SetControlTitle 过程和 SetCtitle 过程是一样的。

```
PROCEDURE SetControlTitle
    (theControl: ControlHandle; title: Str255);
```

theControl　　控制部件的一个间接指针，您希望更改该控制部件的标题。

title　　控制部件的新标题。

说明

SetControlTitle过程把控制记录的ContrlTitle域更改成给定的字符串，并重新绘制控制部件，控制部件标题使用系统字体。

控制管理器允许按钮、复选框和选项按钮的标题具有多行文字。当指定一个多行文字的标题时，行间应加入ASCII字符代码 $ 0D (回车)。若控制部件是一个按钮，每一行都是水平居中的，行间插入相应于字体的行间隔。(每一行的高度等于上升线和下降线之间的距离加上所使用字体的行间间隔。应确保矩形的总高度比行数乘以行的高度大。) 若控制部件是一个复选框或一个选项按钮，文字就采用适合用户当前语系系统的对齐方式，复选框或选项按钮在其矩形中垂直居中。

当您创建控制部件时，您在控制资源或NewControl函数的title域中指定一个初始标题。为了确定一个控制部件的当前标题，应使用 GetControlTitle 过程。

HideControl

为了使一个控制部件不可见（例如在调整它的大小和位置前），应使用HideControl 过程。

```
PROCEDURE HideControl (theControl: ControlHandle);
```

theControl　控制部件的一个间接指针，您希望隐藏该控制部件。

说明

HideControl过程通过更改控制记录的contrlVis域的值，使指定的控制部件不可见，并从屏幕上取消该控制部件。为了填充原来由控制部件占据的区域，HideControl 使用窗口图形端口的背景图案。它还把控制部件的矩形添加至窗口的更新区域，使得原来被控制部件遮盖的内容重新出现在屏幕上。如果控制部件已经不可见，HideControl 就不起作用。

若要使控制部件可见，您可以使用 ShowControl 过程。

特殊考虑

MoveControl和SizeControl过程都自动地调用HideControl和ShowControl。但是，为了在您调用这两个过程时，控制部件不在屏幕上闪烁，您应该先使用HideControl 使控制部件不可见，直至您完成操纵后，再使用 ShowControl。

亦请参阅

5-39 页的程序清单 5-14 解释了在调整滚动杆设定值和位置前，使用HideControl 的方法。

MoveControl

若要在控制部件的窗口中移动它，您可以使用 MoveControl 过程。

```
PROCEDURE MoveControl (theControl : ControlHandle ; h : Integer ; v : Integer) ;
```

theControl　控制部件的一个间接指针，您希望移动该控制部件。

h　控制部件矩形左上角的新位置的水平坐标（以控制部件所在窗口中的局部坐标系表示）。

v　控制部件矩形左上角的新位置的垂直坐标（以控制部件所在窗口中的局部坐标系表示）。

说明

MoveControl 过程把控制部件移至由 h 和 w 参数指定的新位置，用 h 和 w 更改由控制部件的控制记录中的 contrlRect 域指定的矩形。当控制部件可见时，MoveControl 首先隐藏它，然后在新位置重新绘制它。

例如，如果用户调整包含滚动杆的文稿窗口大小，您的应用程序可以使用 MoveControl 把滚动杆移至它的新位置。

亦请参阅

5-67 页的程序清单 5-24 解释了使用 MoveControl 更改滚动杆的位置的方法。

SizeControl

若要更改控制部件矩形的大小，应使用 SizeControl 过程。

```
PROCEDURE SizeControl (theControl: ControlHandle; h: Integer;
v: Integer);
```

theControl	控制部件的一个间接指针，您希望调整该控制部件的大小。
w	调整大小的控制部件的新宽度（以像素为单位）。
h	调整大小的控制部件的新高度（以像素为单位）。

说明

SizeControl 过程更改在控制部件的控制记录的 contrlRect 域中所指定的矩
该过程按照 w 和 h 参数指定的宽度和高度调整矩形的右下角；左上角的位置不变。对于当前可见的控制部件，该过程首先隐藏它，然后按新的大小重新绘制它。SizeControl 过程使用 HideControl，它会更改窗口的更新区域。

亦请参阅

5-67 页的程序清单 5-24 解释了使用 SizeControl 更改一个滚动杆大小的方法。

HiliteControl

如果您需要更改一个控制部件的醒目显示状态，可以使用 HiliteControl 过程。

```
PROCEDURE HiliteControl (theControl: ControlHandle;  hiliteState:
Integer);
```

theControl　　控制部件的一个间接指针。

hiliteState　　从 0 到 255 的一个值，它表明控制部件的醒目显示状态。值 0 表示对有效控制部件为无效状态醒目显示。1 到 253 之间的值表示控制部件醒目显示部分的部分代码。(在5-89 页对 FindControl 的说明中，对部分代码作了解释。) 值 255 标示该控制部件将被设定成无效并应相应地进行绘制。

说明

HiliteControl 过程调用控制部件定义函数重新绘制控制部件，该控制部件的醒目显示状态由 hiliteState 参数指定。HiliteControl 过程使用该参数中的值，更改控制部件的控制记录的 contrlHilite 域的值。

除了滚动杆以外（您应该用 HideControl 过程隐藏它），当控制部件所在的窗口不是最前面的窗口时，您应该使用 HiliteControl 过程使所有的控制部件无效。TrackControl 函数在适当的时候，自动使用 HiliteControl 过程 当您使用TrackControl 时，不需要调用 HiliteControl。

特殊考虑

不要给 hiliteState 参数传送值 254，这个值留作以后使用。

亦请参阅

本书的“对话管理器”一章提供了几个使用 HiliteControl 的实例。

DragControl

如果当用户拖拽一个控制部件时，您需要绘制及移动它的轮廓线或它的指示符（例如滚动杆的滚动块），就可以使用 DragControl 过程。

```
PROCEDURE DragControl (theControl: ControlHandle;
                       startPt: Point;
                       limitRect: Rect; slopRect: Rect;
                       axis: Integer);
```

theControl　　要拖拽的控制部件的一个间接指针。

startPt　　用控制部件所在窗口的局部坐标表示的光标位置，这个位置是用户首次按下鼠标按钮时光标的位置。

limitRect 一个矩形，它在正常情况下应该和窗口内容区一致或包含在窗口内容区之内，这个矩形界定了用户可以拖拽控制部件轮廓线的面积。

slopeRect 一个矩形，它给用户稍大一些的空间移动鼠标，而使控制部件仍然限制在 limitRect 参数指定的矩形中。

axis 用户可以沿着它拖拽控制部件的轴。您可以使用下面的列表给出的常数（及它们代表的值）约束控制部件沿一个轴移动。

常数

```
noConstraint      = 0; {no constraint}
hAxisOnly         = 1; {drag along horizontal axis only}
vAxisOnly         = 2; {drag along vertical axis only}
```

说明

DragControl 过程按照光标的移动，使控制部件的虚线轮廓线在屏幕上移动，直至用户释放鼠标按钮。当用户释放鼠标按钮时，DragControl 调用 MoveControl。MoveControl 再把控制部件移至用户拖拽到的新位置。

TrackControl 函数在适当的时候自动调用 DragControl 过程；当您使用 TrackControl 时，不需要调用 DragControl。

startPt、limitRect、slopeRect 及 axis 参数的意义和窗口管理器的函数 DragGrayRgn 中的相应参数是一样的。

特殊考虑

DragControl 在跟踪光标前，先调用控制部件定义函数。如果您定义了自己的控制部件定义函数，您可以指定自定义的拖拽行为。

汇编语言信息

DragControl 和 TrackControl 一样，请求调用窗口管理器的函数 DragGrayRgn，所以您可以使用全局变量 DragHook 和 DragPattern。

亦请参阅

有关创建您自己的控制部件定义函数的信息，请参阅 5-109 页的“定义您自己的控制部件定义函数”。有关 startPt、limitRect、slopeRect 和 axis 参数更完全的讨论，请参阅本书的“窗口管理器”一章中对 DragGreyRgn 函数的说明，这些参数和 DragControl 函数中使用的完全一样。

SetControlColor

如果您在绘制控制部件时要使用的色彩不同于系统软件使用的系统设定色彩，可以使用SetControlColor过程。SetControlColor过程和SetCtlColor过程是一样的。

```
PROCEDURE SetControlColor (theControl: ControlHandle;
                           newColorTable: CCTabHandle);
```

theControl　控制部件的一个间接指针，您希望更改该控制部件的色彩。

newColorTable　控制部件色彩表记录的一个间接指针。

说明

SetControlColor过程更改指定控制部件的色彩表。如果控制部件当前没有辅助控制记录，SetControlColor将创建一个记录，它包括在参数newColorTable中指定的控制部件色彩表记录，SetControlColor还把辅助控制记录添加至辅助控制列表的头部。如果控制部件已经有了辅助控制记录，SetControlColor用参数newColorTable中指定的控制部件色彩表记录的内容替换辅助记录的色彩表。

为了对控制部件使用非标准的彩色，您必须创建一个控制部件色彩表，这可以通过创建一个彩色控制表记录并调用SetControlColor完成，或者创建一个控制部件色彩表资源完成。在一般情况下，当您使用NewControl创建一个控制部件并希望对它使用非标准彩色时，应使用SetControlColor，或者是在您创建了控制部件后，希望更改它的彩色时使用它。当您希望对在一个控制（'CNTL'）资源中创建的控制部件使用非标准彩色时，您应该创建一个和控制资源具有同样资源标识符 的控制部件色彩表（'cctb'）资源。

您用SetControlColor设定其彩色的控制部件的初始状态应该是不可见的。在使用SetControlColor设定控制部件的色彩后，可使用ShowControl过程使控制部件可见。

特殊考虑

在彩色显示器上，控制管理器自动绘制控制部件，使它们符合系统软件使用的控制部件的彩色。您应该了解，控制部件使用的非标准彩色开始可能会使您的用户感到迷惑。

当您创建一个控制部件色彩表记录后，如果另一个控制部件仍然在使用它，您的应用程序不应该取消分配给它的内存空间。

SetControlAction

如果当您使用 TrackControl 时，为动作过程设定 Pointer (-1)，您就可以使用 SetControlAction 过程设定或更改动作过程。SetControlAction 和 SetCtlAction 是一样的。

```
PROCEDURE SetControlAction (theControl: ControlHandle;
                            actionProc: ProcPtr);
```

theControl　控制部件的一个间接指针，您希望更改它的动作过程。

actionProc　动作过程的一个指针，这个动作过程定义了当用户按住鼠标按钮时，您的应用程序将采取什么动作。

说明

SetControlAction 过程把控制部件的控制记录的 contrlAction 域更改成指向在 actionProc 参数中指定的动作过程。如果用户按下鼠标按钮的同时，光标在指定的控制部件中，TrackControl 就调用这个动作过程。您必须提供这个动作过程，而且只要用户按住鼠标按钮，该动作过程必须定义某个重复执行的动作。(TrackControl 函数总是适当地醒目显示并拖拽控制部件的。)

特殊考虑

只有当您为 TrackControl 的动作过程设定 Pointer (-1) 时，TrackControl 才能使用控制部件的控制记录的 contrlAction 域的值。

一个动作过程通常是在 TrackControl 的一个参数中指定的；在一般情况下您不需要调用 SetControlAction 更改它。

亦请参阅

5-115 页的“定义您自己的动作过程”一节中说明了动作过程。

确定控制部件的值

当您的应用程序创建一个控制部件时，它设定该控制部件的各种值，例如，当前设定值、最大设定值、最小设定值、标题、参考值和动作过程。但是，当用户点按一个控制部件时，您的应用程序常常需要确定该控制部件的当前设定值和其它可能的值。例如当用户点按一个复选框时，您的应用程序必须先确定复选框是否被打了检查标记，然后才能决定在复选框中应该绘制检查标记还是取消检查标记。

您可以分别使用GetControlValue、GetControlTitle、GetControlMinimum、GetControlMaximum、GetControlAction和GetControlReference例程确定一个控制部件当前设定值、标题、最小设定值、最大设定值、动作过程和参考值。若要获得控制部件辅助控制记录的一个间接指针，您可以使用GetAuxiliaryControlRecord函数。您的应用程序可以把辅助控制记录的acRefCon域用于任何目的。若要确定控制部件定义函数中对一个特定的控制部件指定的变量代码，您可以使用GetControlVariant函数。本节还包括了对SetControlReference过程的一个说明，该过程允许您的应用程序更改它对一个控制部件的参考值。

GetControlValue

若要确定一个控制部件的当前设定值，应使用GetControlValue函数。GetControlValue函数和GetCtlValue是一样的。

```
FUNCTION GetControlValue (theControl: ControlHandle): In
                         teger;
```

theControl　　控制部件的一个间接指针。

说明

GetControlValue返回指定控制部件的当前设定值作为它的函数结果，该设定值储存在控制记录的contrlValue域中。

当您创建一个控制部件时，您在控制资源中或在 NewControl 函数的value参数中指定一个初始设定值。您可以使用 SetControlValue 过程更改这个设定值。

亦请参阅

5-37 页的程序清单 5-12 和 5-38 页的程序清单 5-13 分别说明了如何使用GetControlValue确定弹出式菜单和复选框的当前设定值。5-41页的程序清单 5-16、5-53 页的程序清单 5-18 和 5-61 页的程序清单 5-20 说明了如何使用这个函数确定滚动杆的当前设定值。

GetControlMinimum

若要确定一个控制部件的最小设定值，应使用GetControlMinimum函数。GetControlMinimum 函数和GetCtlMin是一样的。

```
FUNCTION GetControlMinimum (theControl: ControlHandle):
                           Integer;
```

the Control　　控制部件的一个间接指针，您希望确定该控制部件的最小设定值。

说明

GetControlMinimum返回指定控制部件的最小设定值作为它的函数结果，该设定值储存在控制记录的 contrlMin 域中。

当您创建一个控制部件时，您在控制资源中或在NewControl 函数的min 参数中指定一个初始最小设定值。您可以使用SetControlMinimum过程更改这个最小设定值。

GetControlMaximum

若要确定一个控制部件的最大设定值，应使用GetControlMaximum函数。GetControlMaximum 函数和 GetCtlMax 是一样的。

```
FUNCTION GetControlMaximum (theControl: ControlHandle): Integer;
```

the Control　　控制部件的一个间接指针，您希望确定该控制部件的最大设定值。

说明

GetControlMaximum返回指定控制部件的最大设定值作为它的函数结果，该设定值储存在控制记录的 contrlMax 域中。

当您创建一个控制部件时，您在控制资源中或在NewControl 函数的max 参数中指定一个初始最大设定值。您可以使用SetControlMaximum过程更改这个最大设定值。

亦请参阅

5-41 页的程序清单 5-16 和 5-61 页的程序清单 5-20 说明了如何使用 GetControlMaximum 确定滚动杆的最大滚动距离。

GetControlTitle

若要确定一个控制部件的标题，应使用 GetControlTitle 函数。GetControlMaximum 函数和 GetCTitle 是一样的。

```
PROCEDURE GetControlTitle (theControl: ControlHandle; VAR title: Str255);
```

theControl　　控制部件的一个间接指针，您希望确定该控制部件的标题。

title　　控制部件的标题。

说明

GetControlTitle 返回指定控制部件的标题，该设定值储存在控制记录的 contrlTitle 域中。

当您创建一个控制部件时，您在控制资源中或在 NewControl 函数的 title 参数中指定一个初始标题。您可以使用 SetControlTitle 过程更改这个标题。

GetControlReference

若要确定一个控制部件的当前参考值，应使用 GetControlReference 函数。GetControlReference 函数和 GetCRefCon 是一样的。

```
FUNCTION GetControlReference (theControl: ControlHandle): LongInt;
```

theControl　控制部件的一个间接指针，您希望确定该控制部件的当前参考值。

GetControlReference　返回指定控制部件的当前参考值作为它的函数结果

说明

当您创建一个控制部件时，您在控制资源中或在 NewControl 函数的 refCon 参数中指定一个初始参考值。参考值储存在控制记录的 contrlRfCon 域中。您可以把这个域用于任何目的，您还可以使用下面说明的 SetControlReference 过程更改这个值。

SetControlReference

若要更改一个控制部件的当前参考值，应使用 SetControlReference 过程。SetControlReference 过程和 SetCRefCon 是一样的。

```
PROCEDURE SetControlReference (theControl: ControlHandle; data:
LongInt);
```

theControl　控制部件的一个间接指针，您希望更改该控制部件的当前参考值。

data　控制部件的新的参考值。

说明

SetControlReference把控制部件的参考值设定为您在data参数中指定的值。

当您创建一个控制部件时，您在控制资源中或在NewControl函数的refCon参数中指定一个初始参考值。该参考值储存在控制记录的contrlRfCon域中。您可以使用GetControlReference函数确定这个当前值。您可以把这个值用于任何目的。

GetControlAction

动作过程的指针储存在控制部件的控制记录的contrlAction域中，若要获得该指针，应使用GetControlAction函数。GetControlAction和GetCtlAction函数是一样的。

```
FUNCTION GetControlAction (theControl: ControlHandle): ProcPtr;
```

theControl　　控制部件的一个间接指针。

说明

GetControlAction返回储存在控制部件的控制记录的contrlAction域中的值作为其函数结果。如果您把TrackControl的actionProc参数设定为Pointer(-1)，这个域就指定了它使用的动作过程。如果用户按下鼠标按钮的同时，光标在控制部件中，该动作过程应定义对此响应的动作。您可以使用SetControlAction过程更改这个动作过程。

亦请参阅

有关定义一个动作过程的信息，请参阅 5-115 页的“定义您自己的动作过程”。

GetControlVariant

若要对于一个特定的控制部件确定控制部件定义函数中指定的变量代码，您可以使用GetControlVariant函数。GetControlVariant和GetCVariant函数是一样的。

```
FUNCTION GetControlVariant (theControl: ControlHandle): Integer;
```

the Control　控制部件的一个间接指针，您希望确定该控制部件的变量代码。

说明

GetControlVariant 返回指定控制部件的变量代码作为它的函数结果。

亦请参阅

变量代码已在 5-14 页的”控制部件定义函数“一节中说明。

GetAuxiliaryControlRecord

应使用GetAuxiliaryControlRecord函数获得控制部件的辅助控制记录的一个二级指针。GetAuxiliaryControlRecord 和 GetAuxCtl 函数是一样的。

```
FUNCTION GetAuxiliaryControlRecord (theControl: ControlHandle;
                                     VAR acHndl: AuxCtlHandle): Boolean;
```

theControl　　控制部件的一个间接指针。

acHndl　　控制部件的辅助控制记录的一个间接指针。

说明

GetAuxiliaryControlRecord 函数在它的 acHndl 参数中返回指定控制部件的辅助控制记录的一个间接指针。除非您需要辅助控制记录的 acRefCon 域（您的应用程序可以将它用于任何目的），您的应用程序一般并不需要存取辅助控制记录。

GetAuxiliaryControlRecord作为函数结果返回的值取决于控制部件的控制色彩表，情况如下:

- 如果您的应用程序对给定的控制部件已经更改了系统设定控制部件色彩表（通过使用 SetControlColor 过程或创建自己的控制部件色彩表），该函数就返回 TRUE
- 如果您的应用程序还没有更改系统设定控制部件色彩表，该函数就返回 FALSE
- 如果您把参数 theControl 设定为NIL，对话管理器就保证了控制部件使用系统设定控制色彩表，GetAuxiliaryControlRecord 返回 TRUE

取消控制部件

当您使用窗口管理器的过程 DisposeWindow 和 CloseWindow 删除窗口时，它们自动地删除所有和该窗口有关的控制部件，并释放这些控制部件占据的存储器。

当您在需要保留的窗口中不再需要一个控制部件时，可以使用 DisposeControl 过程从窗口的控制部件列表中删除该控制部件，并释放它占据的存储器。您可以使用 KillControl 过程一次处理窗口的所有控制部件。

DisposeControl

若要从您需要保留的窗口中删除一个特定的控制部件，应使用 DisposeControl 过程。

```
PROCEDURE DisposeControl (theControl: ControlHandle);
```

theControl　　您希望删除的控制部件的一个间接指针。

说明

DisposeControl 过程从屏幕上删除指定的控制部件，也从窗口的控制部件列表中删除它，该过程还释放控制记录和所有与所该控制部件有关的数据结构占据的存储器。

特殊考虑

窗口管理器的过程 CloseWindow 和 DisposeWindow 自动处理和给定窗口有关的所有控制部件。

亦请参阅

若要删除一个窗口中所有的控制部件，应使用下面说明的 KillControls 过程。CloseWindow 和 DisposeWindow 过程在本书的“窗口管理器”一章中说明。

KillControls

若要在一个您需要保留的特定窗口中删除所有的控制部件，应使用 KillControls 过程。

```
PROCEDURE KillControls (theWindow: WindowPtr);
```

theWindow 窗口的一个指针，该窗口包含要删除的控制部件。

说明

KillControls 过程通过对每个控制部件调用 DisposeControl 过程，处理指定窗口的所有控制部件。

特殊考虑

窗口管理器的过程 CloseWindow 和 DisposeWindow 自动处理给定窗口的所有控制功能。

亦请参阅

本书的“窗口管理器”一章对 CloseWindow 和 DisposeWindow 过程作了说明。

应用程序定义的例程

本节介绍如何创建您自己的控制部件定义函数—这里命名为MyControl,在定义新的、非标准的控制部件时，您的应用程序需要提供自己的控制部件定义函数。本节还介绍了动作过程——这里称为MyAction和MyIndicatorAction，它们定义了附加的动作，每当用户按住鼠标按钮的同时，并且光标在一个控制部件中，这些动作就要被重复地执行。例如，您需要定义一个滚动文稿的动作过程，只要用户按下鼠标按钮时，光标在一个滚动箭头上，就要执行这个动作过程。

定义您自己的控制部件定义函数

除了标准控制部件（按钮、复选框、选项按钮、弹出式菜单和滚动杆）外，控制管理器允许您定义适合于您的应用程序的新的非标准控制部件。例如，您可以定义一个三重选择开关、类似温度计的存储器空间指示符，或者宇宙飞船模拟器的一个推进器控制部件。控制部件和它们的指示符可以占据 QuickDraw 允许的任意形状的区域。

为了定义自己的控制类型，您需要编写一个控制部件定义函数，把它作为 'CDEF' 资源类型进行编译，并储存在您的资源文件中。[有关创建资源更详细的信息，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh : More Macintosh Toolbox*) 一书的“资源管理器”一章。] 每当您创建一个控制部件时，您需要指定一个控制部件定义标识符，控制管理器使用它确定控制部件定义函数。控制部件定义标识符是一个整数，它的高端 12 位是控制部件定义函数的资源标识符，低端 4 位是一个变量代码。这样，对于一个给定的资源标识符和变量代码：

控制部件定义标识符 = 16 * 资源标识符 + 变量代码

例如，按钮、复选框和选项按钮都有标准控制部件定义函数，其资源标识符为 0。因为它们的变量代码分别为 0、1 和 2，它们各自的控制部件定义标识符分别为 0、1 和 2。

您可以定义自己的变量代码，它们不同于控制管理器例程传送给您的控制定义函数的变量代码。这就使您可以使用一个 'CDEF' 资源处理同一个一般控制部件的几种变形。

控制管理器调用资源管理器存取您的具有给定资源标识符 的控制部件定义函数。资源管理器将您的控制部件定义函数读入存储器，并返回该函数的一个间接指针。控制管理器把这个间接指针储存在控制记录的 contrlDefProc 域中。在 24 位地址模式中，变量代码放置在这个域的高位字节；在 32 位地址模式中，变量代码放置在控制部件的 AuxCtlRec 记录的 acReserved 域的最重要字节中。以后，当各种控制管理器例程需要对控制部件执行依赖于类型的动作时，它们调用您的控制部件定义函数，并把变量代码作为参数传送给它。

如果您创建一个控制部件定义函数，您可以使用任意大小的控制部件色彩表记录，并以您希望的方式定义它们的内容，但是 1 到 127 的部分代码要保留给系统定义使用。注意，在这种情况下，您应该对创建的每个控制部件分配明确的辅助记录。

MyControl

如果您希望为应用程序定义新的非标准的控制部件，您必须编写控制部件定义函数，并把它储存在一个资源类型为 'CDEF' 的资源文件中。下面是您宣布一个名为 MyControl 过程的方式：

```
FUNCTION MyControl (varCode: Integer; theControl: ControlHandle;
                    message: Integer; param: LongInt): LongInt;
```

varCode　该控制部件的一个变量代码。为了获得控制部件的控制部件定义标识符，把包含该控制定义函数的 'CDEF' 资源的资源标识符和 16 相乘，把乘积和这个值相加。变量代码允许您在同一基本控制部件的一个 'CDEF' 资源中指定几个控制部件定义标识符。

theControl 操作将要影响的控制部件的一个间接指针。

message　下面列表中的一个值，它指定了您的应用程序必须进行何种操作：

```
常数 drawCntl     = 0; {draw the control or its part}
     testCntl     = 1; {test where mouse button }
                       { is pressed}
     calcCRgns    = 2; {calculate region for }
                       { control or indicator in }
                       { 24-bit systems}
     initCntl     = 3; {peform any additional }
```

```
                          { control initialization}
dispCntl       = 4; {perform any additional }
                    { disposal actions}
posCntl        = 5; {move indicator and }
                    { update its setting}
thumbCntl      = 6; {calculate parameters for }
                    { dragging indicator}
dragCntl       = 7; {perform any custom dragging }
                    { of control or its indicator}
autoTrack      = 8; {execute action procedure }
                    { specified by your function}
calcCntlRgn    = 10;{calculate region for control}
calcThumbRgn   = 11;{calculate region for }
                    { indicator}
```

param　一个值，它的意义取决于在message参数中指定的操作。

说明

控制管理器在各种情况下调用您的控制部件定义函数；控制管理器使用message参数通知您的控制部件定义函数它必须执行的动作。控制管理器向param参数中传送的数据、您的控制部件定义函数必须执行的动作以及您的控制部件定义函数返回的函数结果都取决于控制管理器向message域传送的值。本节的其余部分介绍如何响应控制管理器向message域传送的各种值。

绘制控制部件或它的控制部分

当控制管理器向message参数传送常数 drawCntl 的值时，param参数的低位字具有下列之一的值：

- 值 0 表示整个控制部件
- 值 129 标示必须移动的一个指示符
- 任何其它值，它标示控制部件的一个部分代码（不要使用部分代码 128 和 129，前者保留在将来使用，后者被控制管理器用来标示必须移动的一个指示符。）

附注:

对于drawCntl，param参数的高位字可能包含未定义的数据，故只应该判断这个参数的低位字。

如果指定的控制部件是可见的，您的控制部件定义函数应该在该控制部件的矩形内绘制它（或者在param参数中指定的它的一部分）。如果控制部件不可见，（即它的contrlVis域被设定为 0），那么您的控制部件定义函数不需要作任何事情。

当绘制控制部件或它的控制部分时，要考虑它的控制记录的 contrlHilite 和 contrlValue 域的当前值。

例如，如果控制部件指示符的部分代码被传送给 param 参数（假定该指示符还没有移动），控制管理器就可能调用您的控制部件定义函数，这样您只需要简单地突出显示指示符。但是，当您的应用程序调用 SetControlValue、SetControlMinimum 和 SetControlMaximum 过程时，它们都可能再调用您的控制部件定义函数，重新绘制该指示符。这是因为这些例程无法确定您为指示符选择的部分代码，它们都向 param 传送 129，表示您应该移动该指示符。您的控制部件定义函数必须作为特殊情况检测这个部分代码，并在绘制该指示符之前，把它从原来的位置删除掉。如果您的控制部件具有一个以上的指示符，您应该把 129 解释为所有的指示符。

当向 message 参数传送常数 drawCntl 的值时，您的控制部件定义函数应该总是返回 0 作为它的函数结果。

测试鼠标按下事件发生的地点

为了要求您的控制部件定义函数确定一个指定点是否在一个可见控制部件中，FindControl 函数向 message 参数传送常数 testCntl。这时，param 参数指定了如下的一个点（在控制部件所在窗口的局部坐标系中）：

- 长整数的高位字是该点的垂直坐标
- 长整数的低位字是该点的水平坐标

当向 message 参数传送常数 testCntl 时，您的控制部件定义函数应该返回包含指定点的控制部分的部分代码；如果这个点在控制部件之外，或控制部件是无效的，它就应该返回 0。

计算控制部件和指示符的区域

当控制管理器向 message 参数传送常数 calcCRgns 的值时，您的控制部件定义函数应该计算控制部件或者它的指示符所占据的区域。控制管理器向 param 参数传送一个 QuickDraw 区域的间接指针；您要计算的就是这个区域。如果 param 的高位字被设定，要求的是控制部件的指示符的区域；否则要求的是整个控制部件的区域。在对区域进行计算前，您的控制部件定义函数应该清除该区域间接指针的高位字。

当控制管理器向 message 参数传送常数 calcCntlRgn 的值时，您的控制定义函数应该对指定的控制部件计算传送到 param 参数中的区域。当控制管理器传送常数 calcThumbRgn 时，应计算指示符占据的区域。

当传送常数 calcCRgns、calcCntlRgn 和 calcThumbRgn 时，您的控制部件定义函数总是应该返回 0，并且以控制部件所在窗口的局部坐标系表示计算的区域。

重要附注:

当运行 24 位存储器管理器时，控制管理器传送常数 calcCRgns。当运行 32 位内存管理器时，控制管理器传送的是常数 calcCntlRgn 或 calcThumbRgn。您的控制部件定义函数应该对所有 3 个常数都响应。

执行任何附加的初始化

您的控制部件定义函数可能需要执行依赖于类型的初始化；因此，在创建新的控制功能，适当初始化控制记录的域后，控制管理器就向 message 参数传送常数 initCntl，给您的控制部件定义函数这样一个机会。例如，如果您要执行控制定义函数中的控制部件的动作过程，您需要在控制部件的控制记录的 contrlAction 域中储存 Pointer (-1)。然后，在为这个控制部件调用 TrackControl 时，您将向 TrackControl 的参数 actionProc 传送 Pointer (-1)。

滚动杆的标准控制部件定义函数对包容滚动块的区域分配空间，并在新的控制记录的 contrlData 域中储存该区域的间接指针。

当向 message 参数传送常数 initCntl 时，您的控制部件定义函数应该忽略 param 参数并返回 0 作为函数结果。

执行任何附加的处理动作

DisposeControl 过程向 message 参数传送 dispCntl，以便给您的控制部件定义函数在处理控制部件时，执行任何附加的动作的机会。例如，滚动杆的标准控制定义函数释放滚动块区域占据的存储器，该区域的间接指针是储存在控制部件的控制记录的 contrlData 域中的。

当向 message 参数传送传送常数 dispCntl 的值时，您的控制部件定义函数应该忽略 param 参数，并返回 0 作为函数结果。

移动指示符

当在一个控制部件的指示符中发生释放鼠标事件时，TrackControl 函数调用您的控制部件定义函数并向 message 参数传送 posCntl。这时，param 参数包含一个点（以控制部件所在窗口的局部坐标表示），它指定了垂直和水平的偏移（以像素为单位）您的控制部件定义函数应该按照这个点从指示符的当前位置移动它。通常，这就是用户拖拽指示符时按下和释放鼠标按钮的点之间的偏移。偏移点如下指定：

- param 参数的高位字是该点的垂直偏移量
- param 参数的低位字是该点的水平偏移量

您的控制部件定义函数应该基于给定的偏移量计算控制部件的新设定值，然后按照新的设定值重新绘制控制部件并更新控制记录中的contrlValue域。您的控制定义函数应该忽略param参数并返回 0 作为函数结果。

注意，SetControlValue、SetControlMinimum和SetControlMaximum过程不传送消息posCntl来调用您的控制部件定义函数，它们传送的是消息drawCntl。

计算拖拽指示符的参数

当控制管理器向message参数传送thumbCntl的值时，您的控制部件定义函数的响应应该是计算一个约束拖拽指示符的方式的值（类似于DragControl的limitRect、slopeRect和axis参数）。param参数包含了下列数据结构的一个指针：

```
RECORD
    limitRect,slopRect:          Rect;
    axis:                        Integer;
 END;
```

param^.limitRect.topLeft域在输入上包含了鼠标按下事件首次发生的点。您的控制部件定义函数应该把适当的值储存在param指向的记录的域中；它们类似于窗口管理器的函数DragGreyRgn的具有相同名称的参数。

执行自定拖拽

控制管理器向message参数传送dragCntl，给您的控制部件定义函数一个机会来指定它自己的拖拽控制部件（或它的指示符）的方法。

param参数指定用户是拖拽一个指示符还是拖拽整个控制部件：

- 值 0 表示用户拖拽整个控制部件
- 任何非 0 值表示用户只拖拽指示符

如果您要使用控制管理器的系统设定拖拽方法（即调用DragControl拖拽控制部件，或调用窗口管理器的函数DragGreyRgn拖拽它的指示符），就返回 0 作为您的控制部件定义函数的函数结果。

如果您的控制部件定义函数返回任何非 0 结果，控制管理器将不拖拽您的控制部件，您的控制部件定义函数必须按照光标的移动拖拽指定的控制部件（或它的一部分），直至用户释放鼠标按钮：

- 如果用户拖拽整个控制部件，您的控制部件定义函数应该使用MoveControl过程，在用户释放鼠标按钮后把控制部件放置到它的新位置

- 如果用户拖拽指示符，您的控制部件定义函数必须计算控制部件的新设定值（根据用户在拖拽指示符中按下和释放鼠标按钮时光标所在的两个点之间的偏移像素数决定），然后按照新的设定值重新绘制控制部件，并更新控制记录中的 contrlValue 域。注意，在这种情况下，无论用户是否更改指示符的位置，TrackControl 函数总是返回 0 。这样，您必须确定用户是否更改了控制部件的设定值。您可以通过比较调用 TrackControl 前后的控制部件的值做到这一点

执行一个动作过程

您可以设计一个控制部件，它的动作过程是由您的控制部件定义函数指定的。当您创建该控制部件时，您的控制部件定义函数首先必须响应 initCntl（即先进行初始化动作），在控制部件的控制记录的 contrlAction 域中储存 Pointer (-1)。(前面已经说明过，控制管理器在初始化新的控制记录的域后，向您的控制部件定义函数发送消息 initCntl。）然后当您的应用程序向 TrackControl 函数的 actionProc 域传送 Pointer (-1) 时，TrackControl 用消息 autoTrack 调用您的控制部件定义函数。param 参数指定了发生鼠标按下事件的控制部件的 partCode。然后您的控制定义函数应该使用这个信息，像动作过程一样进行响应。

附注：

对于消息 autoTrack，param 参数的高位字可能包含未定义的数据，因此，您应该只鉴定该参数的低位字。

汇编语言信息

该函数的入口指针必须在首端。

亦请参阅

5-90 页说明了 TrackControl 函数；创建一个动作过程在下节中说明。

定义您自己的动作过程

当控制部件中发生鼠标按下事件时，只要用户按住鼠标按钮，TrackControl 函数通过醒目显示控制部件、拖拽指示符进行适当的响应。您可以定义在这个间隔中重复执行的其它动作。为此，需要定义您自己的动作过程，并在 TrackControl 函数的 actionProc 域中指向这个过程。

当除了指示符以外的控制部分调用您的动作过程时，TrackControl 向您的动作过程传送 (1) 控制部件的一个间接指针，(2) 控制部件的部分代码。然后您的动作过程应该作出适当的响应。

例如，如果用户在一个文字文稿中工作，并当光标处在向下滚动箭头时按下鼠标按钮，您的应用程序应该连续地每次滚动一行文字，直至用户释放鼠标按钮或到达了文稿的尾部。

对于除了指示符以外的控制部分，您应该说明一个动作过程，它接受两个参数：鼠标按下事件在其中发生在其中的控制部件的一个间接指针，和表示发生鼠标按下事件的控制部分的一个整数。下一节中介绍这样的一个动作过程，其名称为 MyAction。

如果鼠标按下事件发生在一个指示符中，您的动作过程不应接受任何参数，因为用户可能把光标拖拽出指示符。5-117 页介绍了这样的一个动作过程，其名称为 MyIndicatorAction。

因为根据鼠标按下事件是否发生在指示符中，调用动作过程需要的参数可以是 0 个或 2 个，您的动作过程只能定义成这两种情况之一。定义您自己的控制定义函数是能够响应控制部件中的所有鼠标按下事件，而不管它是否发生在指示符中的唯一方法。（请参阅 5-109 页的“定义您自己的控制定义函数”一节）。

MyAction

如果您把这个处理指示符以外的控制部分动作过程命名为 MyAction，下面是它的方法介绍：

```
PROCEDURE MyAction (theControl: ControlHandle; partCode: Inte
                    ger);
```

the Control　　鼠标按下事件在其中发生的控制部件的间接指针。

partCode　当光标仍处在首次发生鼠标按下事件的控制部分时，这个参数包含了该控制部分的部分代码。当用户拖拽光标离开原来的控制部分时，该参数值为 0。

说明

您的过程可以对该控制部分执行某个适当的动作。例如，当鼠标按下事件发生在一个滚动箭头或滚动杆的灰度面积时，TrackControl 调用您的动作过程，并传送给它位置信息代码和滚动杆的一个间接指针。您的动作过程应该检验位置信息代码，确定鼠标按下事件发生在控制部件的哪一部分。然后它应该根据情况适当地卷上或卷下一行或一页，然后调用 SetControlValue 过程更改控制部件的设定值并重新绘制滚动块。

汇编语言信息

如果您在全局变量DragHook中储存一个过程的指针，只要用户按住鼠标按钮，该过程就会被重复地调用(没有参数)。TrackControl请求使用窗口管理器的过程 DragGreyRgn，后者又调用DragHook过程。DragGreyRgn函数对于被拖拽的指示符的轮廓线采用储存在全局变量DragPattern中的图案。

亦请参阅

5-59 页的程序清单 5-19 给出了滚动文稿的一对动作过程。还有另外一个把您的动作过程的指针传送给TrackControl的一个参数的方法，您可以使用SetControlAction 过程在控制记录的contrlAction域中储存动作过程的一个指针。当您把Pointer (-1)，而不是过程指针传送给TrackControl时£¨TrackControl使用在控制记录中指向的动作过程。

MyIndicatorAction

如果您把这个关于指示符的动作过程命名为MyIndicatorAction,下面是它的方法介绍:

```
PROCEDURE MyIndicatorAction;
```

说明

您的过程可以执行对于该控制部分适当的任何动作。例如，如果您的应用程序在播放音乐的同时，显示音量控制滑块，您的应用程序应该响应用户对于滑块开关的动作。

亦请参阅

有关的其它考虑，请参阅 5-116 页的过程MyAction。

资源

本节介绍控制（'CNTL'）资源和控制部件色彩表（'cctb'）资源。您可以使用控制资源定义一个控制部件并使用控制部件色彩表资源更改控制部分的系统设定色彩。

控制资源

您可以使用控制资源定义一个控制部件。控制资源是一个 'CNTL' 类型的资源。所有控制资源的资源标识符必须大于 128。利用已定义的控制资源通过GetNewControl 函数（请参阅 5-81 页）创建一个控制部件。控制管理器使用您指定的信息在存储器中创建控制记录。（5-73 页对控制记录进行了说明。）

本节介绍这个资源用 Rez 资源编译程序编译后的结构，Rez 资源编译程序可从 APDA 得到。控制资源的 Rez 输入文件的格式和它编译的输出形式不同，图 5-25 给出了一个说明。如果您只关心如何创建一个控制资源，请参阅 5-15 页的“创建及显示控制部件”一节。

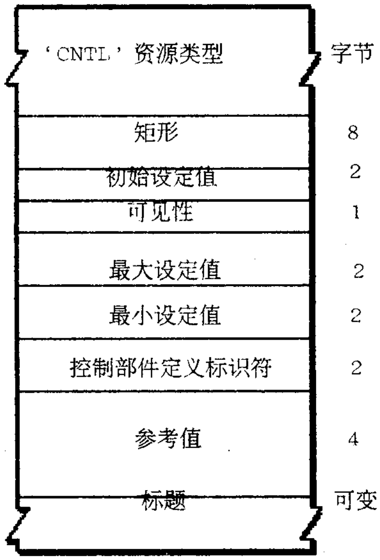


图 5-25 编译的控制（‘CNTL’）资源结构

控制资源编译后的形式包含下列元素：

- 以窗口的局部坐标表示的矩形，它包容控制部件，由于这个矩形包容了控制功能，所以就决定了它的大小和位置
- 控制部件的初始设定值
  - 对于不保持设定值的控制部件，例如按钮，这个值是 0。
  - 对于只有开－关设定值的控制部件，例如复选框和选项按钮，值 0 表示该控制部件的初始状态为关；值 1 表示初始状态为开。
  - 对于设定值可以有一个范围的控制部件，例如滚动杆和拨号，这个元素的初始值应该设定为这个范围中的一个适当的值。

· 对于弹出式菜单，一个组合值指示控制管理器绘制控制部件标题的位置和方式。适当的值以及用来在 Rez 输入文件中指定它们的常数在下面列出：

```
CONST    popupTitleBold      = $00000100;  {boldface font style}
         popupTitleItalic    = $00000200;  {italic font style}
         popupTitleUnderline = $00000400;{underline font }
                                           { style}
         popupTitleOutline = $00000800;  {outline font style}
         popupTitleShadow    = $00001000;  {shadow font style}
         popupTitleCondense= $00002000;  {condensed text}
         popupTitleExtend    = $00004000;  {extended text}
         popupTitleNoStyle = $00008000;  {monostyle text}
         popupTitleLeftJust = $00000000; {place title left }
                                           { of pop-up box}
         popupTitleCenterJust= $00000001;{center title over }
                                           { pop-up box}
         popupTitleRightJust = $000000FF;{place title right }
                                           { of pop-up box}
```

■ 控制部件的可见性。如果这个元素包含 TRUE，GetNewControl 就立即绘制该控制功能，而不使用应用程序对窗口的标准更新机制。如果这个元素包含 FALSE，应用程序必须在准备显示控制部件时使用 ShowControl 过程（请参阅 5-86 页）

■ 填充，它应该设定为 0

■ 控制部件的最大设定值
  · 对于不保持设定值的控制部件，例如按钮，这个值是 1。
  · 对于只有开／关设定值的控制部件，例如复选框和选项按钮，这个元素应该是值 1（表示开的状态）。
  · 对于设定值可以有一个范围的控制部件，例如滚动杆和拨号，这个元素应该是可能的最大值；当应用程序使滚动杆的最大设定值和最小设定值相等时，控制部件定义函数自动使滚动杆无效；当应用程序使最大设定值超过最小设定值时，控制部件定义函数使滚动杆变成有效。

  · 对于弹出式菜单，这个元素包含控制部件标题的宽度（以象素为单位）。

■ 控制部件的最小设定值
  · 对于不保持设定值的控制部件，例如按钮，这个值是 0。
  · 对于只有开／关设定值的控制部件，例如复选框和选项按钮，这个元素应该是值 0（表示关的状态）。
  · 对于设定值可以有一个范围的控制部件，例如滚动杆和拨号，这个元素应该是可能的最小值。
  · 对于弹出式菜单，这个元素包含 'MENU' 资源的资源标识符，'MENU' 资源对菜单命令项目进行了说明。

■ 控制部件定义标识符，控制管理器使用它确定该控制部件的控制部件定义函数。5-109 页的“定义您自己的控制部件定义函数”一节介绍了如何创建控制部件定义函数以及它们相应的控制部件定义标识符。下面的列表给出了标准控制部件的控制部件定义标识符编号，以及在 Rez 输入文件中代表它们的常数

常数

```
popupTitleBold      = $00000100;  {boldface font style}
popupTitleItalic    = $00000200;  {italic font style}
popupTitleUnderline = $00000400;{underline font }
                                  { style}
popupTitleOutline = $00000800;  {outline font style}
popupTitleShadow    = $00001000;  {shadow font style}
popupTitleCondense= $00002000;  {condensed text}
popupTitleExtend    = $00004000;  {extended text}
popupTitleNoStyle = $00008000;  {monostyle text}
popupTitleLeftJust = $00000000; {place title left }
                                  { of pop-up box}
popupTitleCenterJust= $00000001;{center title over }
                                  { pop-up box}
popupTitleRightJust = $000000FF;{place title right }
                                  { of pop-up box}
```

附注:

在正常情况下，按钮、复选框、选项按钮或弹出式菜单的标题以系统字体出现，在 Roman 语系系统中是 12 点的 Chicago 字体。不要使用比 12 点小的字体，某些语系系统，例如 KanjiTalk，要求 12 点的字体。一般情况下，您应该在控制部件中使用系统字体；这样可以方便应用程序的本地化。但是，如果您一定需要用当前窗口图形端口的字体显示控制部件，也可以给弹出式菜单控制定义 ID 添加常数 popupUseWFont，或者给其它标准控制部件定义标识符添加常数 useWFont。

■ 控制部件的参考值，只有应用程序设定并使用它（一个例外情况在 5-25 页中说明过：应用程序把 popupUseAddResMenu 变量代码添加至 popupMenuProc 控制部件定义标识符）

- 对于需要标题的控制部件，例如按钮、复选框、选项按钮和弹出式菜单，该参数为标题的字符串；对于不需要标题的控制部件，则是一个空字符串

在您使用 GetNewControl 创建控制部件后，您可以使用 SetControlValue、SetControlMaximum、SetControlMinimum、SetControlReference 和 SetControlTitle 例程分别更改当前设定值、最大设定值、最小设定值、参考值和标题。您可以使用 MoveControl 和 SizeControl 例程更改控制部件的矩形。您可以使用 GetControlValue、GetControlMaximum、GetControlMinimum、GetControlReference 和 GetControlTitle 例程确定控制部件的值。

控制部件色彩表资源

在彩色显示器上，控制管理器自动地使它绘制的控制部分和系统软件使用的控制功能的色彩匹配。

如果您感到绝对需要使用非标准的色彩，控制管理器也允许您这样做。您的应用程序可以创建一个控制部件色彩表（ 'cctb' ）资源指定您对控制部件要使用的彩色；您必须给这个控制部件色彩表资源和该控制部件的控制（ 'CNTL' ）资源同样的资源标识符（请参阅 5-118 页）。当您调用 GetNewControl 函数创建该控制部件时，控制管理器自动地装入一个控制部件色彩表资源，它和为 GetNewControl 指定的控制资源具有同样的资源标识符。控制管理器还为该控制部件创建一个辅助控制记录；有关辅助控制记录请参阅 5-76 页。

附注：

如果您的控制部件使用非标准的色彩，可能在一开始会使用户感到迷惑。

在一般情况下，您对控制资源中定义的控制部件使用一个控制部件色彩表资源。若要更改控制部件的彩色或在您用NewControl创建的控制部件中使用非标准的彩色，应创建一个控制部件色彩表记录并使用SetControlColor过程。有关控制部件色彩表请参阅 5-77 页£ªSetControlColor 过程请参阅 5-101 页。

控制部件色彩表资源属于 'cctb' 类型。所有控制部件色彩表资源的资源标识符编号必须大于 128。下一页的图 5-26 给出了一个控制部件色彩表资源的格式。注意，DisposeControl 不删除控制部件色彩表资源；因此，您应该使每个控制部件色彩表资源是可清除的。



图 5-26 经过编译的控制部件色彩表（'cctb'）资源的结构

您通过在 'cctb' 资源类型的资源中指定这些元素，定义控制部件色彩表资源:

- 保留。应该总是设定为 0
- 保留。应该总是设定为 0
- 控制部分的数目。对于滚动杆以外的标准控制部件，它应该设定为 3，因为这些控制部件包含框架、控制主体和文字。对于滚动杆，它应该是 12。滚动杆包括框架、主体、滚动块；滚动杆的每个部分有不同的醒目显示状态和着色。若要创建具有更多部分的滚动杆，您必须创建自己定义的控制部件定义函数（请参阅 5-109 页的“定义您自己的控制部件定义函数”一节），它能够识别添加的部分
- 第一部分标识码。标识控制部分彩色的一个值或常数。控制部分标识码可以按任何次序列表。滚动杆的控制部件定义函数可以使用一个以上的控制部分标识码，为滚动杆的每个部分产生真实使用的彩色

常数

```
cFrameColor= 0;     {frame color; for scroll bars, used to produce }
                    { foreground color for scroll arrows & gray area}
cBodyColor  = 1;    {body color; for scroll bars, used to produce }
                    { colors in the scroll box}
cTextColor  = 2;    {text color; unused for scroll bars}
```

```
cArrowsColorLight= 5;  {Used to produce colors in arrows & scroll
                        bar }
                       { background color}
cArrowsColorDark= 6;   {Used to produce colors in arrows & scroll
                        bar }
                       { background color}
cThumbLight     = 7;   {Used to produce colors in scroll box}
cThumbDark      = 8;   {Used to produce colors in scroll box}
cHiliteLight    = 9;   {Use same value as wHiliteColorLight in
                        'wctb'}
cHiliteDark     = 10;  {Use same value as wHiliteColorDark in
                        'wctb'}
cTitleBarLight   = 11; {Use same value as wTitleBarLight in 'wctb'}
cTitleBarDark   = 12;  {Use same value as wTitleBarDark in 'wctb'}
cTingeLight     = 13;  {Use same value as wTingeLight in 'wctb'}
cTingeDark      = 14;  {Use same value as wTingeDark in 'wctb'}
```

- 红色分量。绘制这个控制部分时使用的一个整数，它表示红色分量的强度。在这个和下两个元素中，使用16一位符号整数来给出3种可叠加基色的强度值
- 绿色分量。绘制这个控制部分时使用的一个整数，它表示绿色分量的强度
- 蓝色分量，绘制这个控制部分时使用的一个整数，它表示蓝色分量的强度
- 下一个控制部分的部分标识码和红、绿、蓝分量。您可在这个资源中以任何次序列出各部分。如果应用程序指定一个找不到的部分标识码，控制管理器就对控制部件的第一个可标识部分使用这些彩色。如果某个部分没有在控制部件色彩表中列出，对话管理器就会以其系统设定色彩绘制它

控制部件定义函数

控制部件定义函数的资源类型是 'CDEF'。资源数据是经过编译或汇编的函数的代码。若想了解关于创建一个控制部件定义函数的信息，请参阅从5 - 109页的“定义您自己的控制部件定义函数”。

控制管理器小结

Pascal 小结

常数

```
常数
    {control definition IDs}
    pushButProc         = 0;      {button}
    checkBoxProc        = 1;      {checkbox}
    radioButProc        = 2;      {radio button}
    useWFont            = 8;      {add to above to display control title in }
                                  { the window font}
    scrollBarProc       = 16;     {scroll bar}
    popupMenuProc       = 1008;   {pop-up menu}
    popupMenuCDEFproc = popupMenuProc;  {synonym for compatibility}

    {pop-up menu CDEF variation codes}
    popupFixedWidth     = $0001;  {add to popupMenuProc to use }
                                  { fixed-width control}
    popupUseAddResMenu= $0004;    {add to popupMenuProc to specify a }
                                  { value of type ResType in the }
                                  { contrlRfCon field of the control }
                                  { record; Menu Manager adds }
                                  { resources of this type to the menu}
    popupUseWFont       = $0008;  {add to popupMenuProc to show control }
                                  { title in the window font}

    {part codes}
    inButton            = 10;     {button}
    inCheckBox          = 11;     {checkbox or radio button}
    inUpButton          = 20;     {up arrow for a vertical scroll bar, }
                                  { left arrow for a horizontal scroll bar}
    inDownButton        = 21;     {down arrow for a vertical scroll bar, }
                                  { right arrow for a horizontal scroll bar}
    inPageUp            = 22;     {gray area above scroll box for a }
                                  { vertical scroll bar, gray area to }
                                  { left of scroll box for a horizontal }
                                  { scroll bar}
```

```
inPageDown          = 23;    {gray area below scroll box for a }
                             { vertical scroll bar, gray area to }
                             { right of scroll box for a horizontal }
                             { scroll bar}
inThumb             = 129;   {scroll box (or other indicator)}

{pop-up title characteristics}
popupTitleBold          = $00000100; {boldface font style}
popupTitleItalic        = $00000200; {italic font style}
popupTitleUnderline     = $00000400; {underline font style}
popupTitleOutline       = $00000800; {outline font style}
popupTitleShadow        = $00001000; {shadow font style}
popupTitleCondense      = $00002000; {condensed characters}
popupTitleExtend        = $00004000; {extended characters}
popupTitleNoStyle       = $00008000; {monostyled text}
popupTitleLeftJust      = $00000000; {place title left of pop-up box}
popupTitleCenterJust    = $00000001; {center title over pop-up box}
popupTitleRightJust     = $000000FF; {place title right of pop-up box}

{axis constraints for DragControl procedure}
noConstraint            = 0;     {no constraint}
hAxisOnly               = 1;     {drag along horizontal axis only}
vAxisOnly               = 2;     {drag along vertical axis only}

{constants for the message parameter in a control definition function}
drawCntl            = 0; {draw the control or its part}
testCntl            = 1; {test where mouse button is pressed}
calcCRgns           = 2; {calculate region for control or indicator in}
                         { 24-bit systems}
initCntl            = 3; {peform any additional control initialization}
dispCntl            = 4; {take any additional disposal actions}
posCntl             = 5; {move indicator and update its setting}
thumbCntl           = 6; {calculate parameters for dragging indicator}
dragCntl            = 7; {perform any custom dragging of control or }
                         { its indicator}
autoTrack           = 8; {execute action procedure specified by your }
                         { function}
calcCntlRgn         = 10;{calculate region for control}
calcThumbRgn        = 11;{calculate region for indicator}

{part identifiers for ColorSpec records in a control color table resource}
cFrameColor         = 0; {frame color; for scroll bars, also fore- }
                         { ground color for scroll arrows and gray area}
```

```
cBodyColor     = 1; {for scroll bars, background color for }
                    { scroll arrows and gray area; for other }
                    { controls, the fill color for body of control}
cTextColor     = 2; {text color; unused for scroll bars}
cThumbColor    = 3; {Reserved}
```

数据类型

```
TYPE    ControlPtr          = ^ControlRecord;
        ControlHandle = ^ControlPtr;

        ControlRecord =
        PACKED RECORD
            nextControl:      ControlHandle;{next control}
            contrlOwner:  WindowPtr;        {control's window}
            contrlRect:       Rect;         {rectangle}
            contrlVis:        Byte;         {255 if visible}
            contrlHilite:     Byte;         {highlight state}
            contrlValue:      Integer;      {control's current setting}
            contrlMin:        Integer;      {control's minimum setting}
            contrlMax:        Integer;      {control's maximum setting}
            contrlDefProc:    Handle;       {control definition function}
            contrlData:       Handle;       {data used by contrlDefProc}
            contrlAction:     ProcPtr;      {action procedure}
            contrlRfCon:      LongInt;      {control's reference value}
            contrlTitle:      Str255;       {control's title}
        END;

        AuxCtlPtr           = ^AuxCtlRec;
        AuxCtlHandle   = ^AuxCtlPtr;

        AuxCtlRec =
        RECORD
            acNext:       AuxCtlHandle; {handle to next AuxCtlRec}
            acOwner:      ControlHandle;{handle to this record's control}
            acCTable:     CCTabHandle;  {handle to color table record}
            acFlags:      Integer;      {reserved}
            acReserved:   LongInt;      {reserved for future use}
            acRefCon:     LongInt;      {for use by application}
        END;
```

```
CCTabPtr      = ^CtlCTab;
CCTabHandle   = ^CCTabPtr;

CtlCTab =
RECORD
     ccSeed:  LongInt; {reserved; set to 0}
     ccRider: Integer; {reserved; set to 0}
     ctSize:  Integer; {number of ColorSpec records in next }
                       { field; 3 for standard controls}
     ctTable: ARRAY[0..3] OF ColorSpec;
END;
```

控制管理器例程

```
Creating Controls
 FUNCTION GetNewControl  (controlID: Integer; owner: WindowPtr)
                : ControlHandle;
 FUNCTION NewControl (theWindow: WindowPtr; boundsRect: Rect;
                title: Str255; visible: Boolean; value: Integer;
                min: Integer; max: Integer; procID: Integer; refCon
                : LongInt) : ControlHandle;
```

绘制控制部件

```
{UpdateControls is also spelled as UpdtControl}
 PROCEDURE ShowControl    (theControl: ControlHandle);
 PROCEDURE UpdateControls(theWindow: WindowPtr; updateRgn: RgnHandle);
 PROCEDURE DrawControls   (theWindow: WindowPtr);
 PROCEDURE Draw1Control   (theControl: ControlHandle);
```

处理控制部件中的鼠标单击事件

```
FUNCTION FindControl     (thePoint: Point; theWindow: WindowPtr;
                         VAR theControl: ControlHandle): Integer;
 FUNCTION TrackControl   (theControl: ControlHandle; thePoint: Point;
                         actionProc: ProcPtr): Integer;
 FUNCTION TestControl    (theControl: ControlHandle; thePt: Point)
                         : Integer;
```

更改控制部件的设置和显示状态

```
{some routines have 2 spellings--see Table 5-1 for the alternate spellings}
PROCEDURE SetControlValue     (theControl: ControlHandle; theValue: Integer);
PROCEDURE SetControlMinimum   (theControl: ControlHandle; minValue: Integer);
PROCEDURE SetControlMaximum   (theControl: ControlHandle; maxValue: Integer);
PROCEDURE SetControlTitle     (theControl: ControlHandle; title: Str255);
PROCEDURE HideControl         (theControl: ControlHandle);
PROCEDURE MoveControl         (theControl: ControlHandle; h: Integer;
                              v: Integer);
PROCEDURE SizeControl         (theControl: ControlHandle; w: Integer; h:Integer);
PROCEDURE HiliteControl       (theControl: ControlHandle;
                              hiliteState: Integer);
PROCEDURE DragControl         (theControl: ControlHandle; startPt: Point;
limitRect: Rect;              slopRect: Rect;
                              axis: Integer);
PROCEDURE SetControlColor     (theControl: ControlHandle; newColorTable:
                              CCTabHandle);
PROCEDURE SetControlAction    (theControl: ControlHandle;
                              actionProc: ProcPtr);
```

控制部件的设定值

```
{some routines have 2 spellings--see Table 5-1 for the alternate spellings}
FUNCTION GetControlValue      (theControl: ControlHandle): Integer;
FUNCTION GetControlMinimum    (theControl: ControlHandle): Integer;
FUNCTION GetControlMaximum    (theControl: ControlHandle): Integer;
PROCEDURE GetControlTitle     (theControl: ControlHandle; VAR title: Str255);
FUNCTION GetControlReference
                              (theControl: ControlHandle): LongInt;
PROCEDURE SetControlReference
                              (theControl: ControlHandle; data: LongInt);
FUNCTION GetControlAction     (theControl: ControlHandle): ProcPtr;
FUNCTION GetControlVariant    (theControl: ControlHandle): Integer;
FUNCTION GetAuxiliaryControlRecord
                              (theControl: ControlHandle;
                              VAR acHndl: AuxCtlHandle): Boolean;


Removing Controls
PROCEDURE DisposeControl      (theControl: ControlHandle);
PROCEDURE KillControls        (theWindow: WindowPtr);
```

应用程序定义的过程

用户自定义的控制部件过程

```
FUNCTION MyControl  (varCode: Integer; theControl: ControlHandle;
                    message: Integer;
                    param: LongInt) : LongInt;
```

用户自定义的控制部件响应过程

```
PROCEDURE MyAction(theControl: ControlHandle; partCode: Integer);
PROCEDURE MyIndicatorAction;
```

C 小结

常数

```
enum {
    /*control definition IDs*/
    pushButProc        = 0,          /*button*/
    checkBoxProc       = 1,          /*checkbox*/
    radioButProc       = 2,          /*radio button*/
    useWFont           = 8,          /*add to above to display control */
                                     /* title in the window font*/
    scrollBarProc      = 16,         /*scroll bar*/
    popupMenuProc      = 1008,       /*pop-up menu*/

    /*pop-up menu CDEF variation codes*/
    popupFixedWidth    = 1 << 0,     /*add to popupMenuProc to use */
                                     /* use fixed-width control*/
    popupUseAddResMenu = 1 << 2,     /*add to popupMenuProc to specify a */
                                     /* value of type ResType in the */
                                     /* contrlRfCon field of the control */
                                     /* record; Menu Manager adds */
                                     /* resources of this type to the menu*/
    popupUseWFont      = 1 << 3      /*add to popupMenuProc to display */
                                     /* control title in the window font*/
};
```

```
enum {
    /*part codes*/
    inButton       = 10,    /*button*/
    inCheckBox     = 11,    /*checkbox or radio button*/
    inUpButton     = 20,    /*up arrow for a vertical scroll bar, */
                            /* left arrow for a horizontal scroll bar*/
    inDownButton   = 21,    /*down arrow for a vertical scroll bar, */
                            /* right arrow for a horizontal scroll bar*/
    inPageUp       = 22,    /*gray area above scroll box for a */
                            /* vertical scroll bar, gray area to */
                            /* left of scroll box for a horizontal */
                            /* scroll bar*/
    inPageDown     = 23,    /*gray area below scroll box for a */
                            /* vertical scroll bar, gray area to */
                            /* right of scroll box for a horizontal */
                            /* scroll bar*/
    inThumb        = 129    /*scroll box (or other indicator)*/
};

enum {
    /*pop-up title characteristics*/
    popupTitleBold  = 1 << 8,        /*boldface font style*/
    popupTitleItalic = 1 << 9,       /*italic font style*/
    popupTitleUnderline = 1 << 10,   /*underline font style*/
    popupTitleOutline = 1 << 11,     /*outline font style*/
    popupTitleShadow = 1 << 12,      /*shadow font style*/
    popupTitleCondense = 1 << 13,    /*condensed text*/
    popupTitleExtend = 1 << 14,      /*extended text*/
    popupTitleNoStyle = 1 << 15      /*monostyled text*/
};

enum {
    /*pop-up title characteristics*/
    popupTitleLeftJust = 0x00000000,     /*place title left of pop-up box*/
    popupTitleCenterJust = 0x00000001,   /*center title over pop-up box*/
    popupTitleRightJust = 0x000000FF, /*place title right of pop-up box*/

    /*axis constraints for DragControl procedure*/
    noConstraint        = 0, /*no constraint*/
    hAxisOnly           = 1, /*constrain movement to horizontal axis only*/
    vAxisOnly           = 2, /*constrain movement to vertical axis only*/
```

```
/*constants for the message parameter in a control definition function*/
    drawCntl      = 0,     /*draw the control or control part*/
    testCntl      = 1,     /*test where mouse button was pressed*/
    calcCRgns     = 2,     /*calculate region for control or indicator in */
                           /* 24-bit systems*/
    initCntl      = 3,     /*do any additional control initialization*/
    dispCntl      = 4,     /*take any additional disposal actions*/
    posCntl       = 5,     /*move indicator and update its setting*/
    thumbCntl     = 6,     /*calculate parameters for dragging indicator*/
    dragCntl      = 7,     /*peform any custom dragging of control or */
                           /* its indicator*/
    autoTrack     = 8,     /*execute action procedure specified by your */
                           /* function*/
    calcCntlRgn   = 10,    /*calculate region for control*/
    calcThumbRgn  = 11,    /*calculate region for indicator*/

/*part identifiers for ColorSpec records in a control color table resource*/
    cFrameColor   = 0,     /*frame color; for scroll bars, also foreground */
                           /* color for scroll arrows and gray area*/
    cBodyColor    = 1,     /*for scroll bars, background color for scroll */
                           /* arrows and gray area; for other controls, */
                           /* the fill color for body of control*/
    cTextColor    = 2,     /*text color; for scroll bars, unused*/
    cThumbColor   = 3      /*Reserved*/
};
```

数据类型

```
struct   ControlRecord {
    struct ControlRecord **nextControl; /*next control*/
    WindowPtr     contrlOwner;          /*control's window*/
    Rect          contrlRect;           /*rectangle*/
    unsigned char contrlVis;            /*255 if visible*/
    unsigned char contrlHilite;         /*highlight state*/
    short         contrlValue;          /*control's current setting*/
    short         contrlMin;            /*control's minimum setting*/
    short         contrlMax;            /*control's maximum setting*/
    Handle        contrlDefProc;        /*control definition function*/
    Handle        contrlData;           /*data used by contrlDefProc*/
    ProcPtr       contrlAction;         /*action procedure*/
    long          contrlRfCon;          /*control's reference value*/
    Str255        contrlTitle;          /*control's title*/
    };
```

```
typedef struct ControlRecord ControlRecord;
typedef ControlRecord *ControlPtr, **ControlHandle;

struct   AuxCtlRec {
    Handle         acNext;        /*handle to next AuxCtlRec*/
    ControlHandle  acOwner;       /*handle to this record's control*/
    CCTabHandle    acCTable;      /*handle to color table record*/
    short          acFlags;       /*reserved*/
    long           acReserved;    /*reserved for future use*/
    long           acRefCon;      /*for use by application*/
};

typedef struct AuxCtlRec AuxCtlRec;
typedef AuxCtlRec *AuxCtlPtr, **AuxCtlHandle;

struct CtlCTab {
    long           ccSeed;        /*reserved; set to 0*/
    short          ccRider;       /*reserved; set to 0*/
    short          ctSize;        /*number of ColorSpec records in next */
                                  /* field; 3 for standard controls*/
    ColorSpec      ctTable[4];
    };
typedef struct CtlCTab CtlCTab;
typedef CtlCTab *CCTabPtr, **CCTabHandle;
```

控制管理器例程

```
Creating Controls
pascal ControlHandle GetNewControl
                  (short controlID, WindowPtr owner);
pascal ControlHandle NewControl
                  (WindowPtr theWindow, const Rect *boundsRect,
                  ConstStr255Param title, Boolean visible,
                  short value, short min, short max,
                  short procID, long refCon);

Drawing Controls
/*UpdateControls is also spelled as UpdtControl*/
pascal void ShowControl     (ControlHandle theControl);
pascal void UpdateControls (WindowPtr theWindow, RgnHandle updateRgn);
pascal void DrawControls    (WindowPtr theWindow);
pascal void Draw1Control    (ControlHandle theControl);
```

处理控制部件中的鼠标单击事件过程

```
pascal short FindControl    (Point thePoint, WindowPtr theWindow, ControlHandle
                             *theControl);
pascal short TrackControl   (ControlHandle theControl, Point thePoint, ProcPtr
                             actionProc);
pascal short TestControl    (ControlHandle theControl, Point thePt);

Changing Control Settings and Display
/*some routines have 2 spellings—see Table 5-1 for the alternate spellings*/
pascal void SetControlValue(ControlHandle theControl, short theValue);
pascal void SetControlMinimum
                            (ControlHandle theControl, short minValue);
pascal void SetControlMaximum
                            (ControlHandle theControl, short maxValue);
pascal void SetControlTitle(ControlHandle theControl,
                            ConstStr255Param title);
pascal void HideControl     (ControlHandle theControl)
pascal void MoveControl     (ControlHandle theControl, short h, short v);
pascal void SizeControl     (ControlHandle theControl, short w, short h);
pascal void HiliteControl   (ControlHandle theControl, short hiliteState);
pascal void DragControl     (ControlHandle theControl, Point startPt,
                            const Rect *limitRect,
                            const Rect *slopRect, short axis);
pascal void SetControlAction(ControlHandle theControl, ProcPtr actionProc)
pascal void SetControlColor(ControlHandle theControl,
                            CCTabHandle newColorTable);
```

控制部件设定值

```
/*some routines have 2 spellings—see Table 5-1 for the alternate spellings*/
pascal short GetControlValue
                        (ControlHandle theControl);
pascal short GetControlMinimum
                        (ControlHandle theControl);
pascal short GetControlMaximum
                        (ControlHandle theControl);
pascal void GetControlTitle(ControlHandle theControl, Str255 title);
pascal long GetControlReference
                        (ControlHandle theControl);
pascal void SetControlReference
                        (ControlHandle theControl, long data);
pascal ProcPtr GetControlAction
                        (ControlHandle theControl);
```

```
pascal short GetControlVariant
                         (ControlHandle theControl);
pascal Boolean GetAuxiliaryControlRecord
                         (ControlHandle theControl,
                         AuxCtlHandle *acHndl);
```

删除控制部件例程

```
pascal void DisposeControl (ControlHandle theContr
                            ol);
pascal void KillControls   (WindowPtr theWindow);
```

应用程序定义的例程

用户自定义的控制部件过程

```
pascal long MyControl    (short varCode, ControlHandle
                          theControl,
short message, long param);
```

用户自定义的控制部件响应过程

```
pascal void MyAction     (ControlHandle theControl,
                          short partCode);
pascal void MyIndicatorAction;
```

汇编语言小结

数据结构

ControlRecord 数据结构

0	nextControl	长型	控制列表中下一个控制部件的间接指针
4	contrlOwner	长型	该控制部件的窗口的指针
8	contrlRect	8 字节	控制部件的矩形
16	contrlVis	1 字节	如果控制部件可视其值为 255
17	contrlHilite	1 字节	醒目显示状态
18	contrlValue	字	控制部件的当前设定状态
20	contrlMin	字	控制部件的最小设定状态
22	contrlMax	字	控制部件的最大设定状态
24	contrlDefProc	长型	控制部件定义函数的间接指针
28	contrlData	长型	控制部件定义函数使用的数据
32	contrlAction	长型	动作过程的地址
36	contrlRfCon	长型	控制部件的参考值
40	contrlTitle	256 字节	控制部件标题（首字节为长度）

辅助控制记录数据结构

0	acNext	长型	handle to next AuxCtlRec record in control list
4	acOwner	长型	handle to this record's control
8	acCTable	长型	handle to color table for this control
12	acFlags	字	miscellaneous flags
14	acReserved	长型	reserved for use by Apple Computer, Inc.
18	acRefCon	长型	for use by application

全局变量

AuxCtlHead	First in a linked list of auxiliary control records
AuxWinHead	Contains a pointer to the linked list of auxiliary control records
DragHook	Address of procedure to execute during TrackControl and DragControl
DragPattern	Pattern of dragged region's outline (8 bytes)

第六章 对话管理器

警告和对话框介绍 6－6
警告类型 6－8
对话框类型 6－9
定模式对话框 6－10
可移动定模式对话框 6－11
不定模式对话框 6－12
警告和对话框中的项目 6－13
警告和对话框中的事件 6－14
警告栏、对话框和窗口管理器 6－15
关于对话管理器 6－16
使用对话管理器 6－17
创建示警声和警告栏 6－18
创建对话框 6－23
为警告栏和对话框提供项目 6－26
项目类型 6－30
显示矩形 6－32
启用和停用项目 6－36
项目的资源标识符 6－36
按钮、复选框和选项按钮的标题 6－37
静态文本项目和可编辑文字项目的文字字符串 6－40
弹出式菜单命令项目 6－42
项目间键盘引导 6－44
操纵项目 6－44
更改静态文字 6－46
从可编辑文字项目中获取文字 6－48
在一个现有的对话框中添加项目 6－51
使用一个定义应用程序的项目来绘制一个系统设定按钮的黑体轮廓线 6－56

显示警告栏和对话框 6－61
定位警告栏和对话框 6－62
停用警告栏和定模式对话框后面的窗口 6－64
显示不定模式对话框 6－66
针对定模式对话框调整菜单 6－68
针对可移动定模式和不定模式对话框调整菜单 6－73
显示多个警告栏和对话框 6－74
从后台显示警告栏和对话框 6－74
在您的警告栏和对话框中加入彩色 6－75
处理警告栏和对话框中的事件 6－77
响应控制部件中的事件 6－78
响应可编辑文字项目中的事件 6－79
响应警告栏中的事件 6－81
响应定模式对话框中的事件 6－82
为警告栏和定模式对话框编写事件过滤函数 6－86
响应不定模式和可移动定模式对话框中的鼠标事件 6－90
响应不定模式和可移动定模式对话框中的键盘事件 6－94
响应不定模式和可移动定模式对话框中的激活和更新事件 6－97
关闭对话框 6－100
对话管理器参考资料 6－101
数据结构 6－101
对话记录 6－101
对话管理器例程 6－102
初始化对话管理器 6－102
创建警告 6－105
创建和处理对话框 6－113
操纵警告栏和对话框中的项目 6－120
处理警告栏和对话框中的文字 6－129
处理对话框中的事件 6－135
应用程序定义的例程 6－143
资源 6－147
对话资源 6－148
警告资源 6－150
项目列表资源 6－151
对话项目色彩表资源 6－156
警告色彩表资源 6－157
项目色彩表资源 6－158
对话管理器小结 6－165
Pascal 小结 6－165
常数 6－165
数据类型 6－166
对话管理器例程 6－166
应用程序定义的例程 6－168

C 小结 6 － 168
常数 6 － 168
数据类型 6 － 169
对话管理器例程 6 － 170
应用程序定义的例程 6 － 172
汇编语言小结 6 － 172
数据结构 6 － 172
全局变量 6 － 172

第六章

对话管理器

本章介绍您的应用程序怎样使用对话管理器向用户报告异常情况、发出警告，并向用户征求信息。例如，在一些情况下您的应用程序可能不能正常地执行一个命令；在另一些情况下，在您的应用程序能执行一个命令之前用户必须指定多个参数。针对像这样的情形，Macintosh 用户界面包含下面两个功能：

- **警告**——包括示警声和警告栏——每当在您的应用程序内发生一个异常的或可能不符合需要的情况时它们就会警告用户。
- **对话框**，它允许用户在您的应用程序执行一个命令之前提供额外的信息或修改设定状态。

阅读本章了解如何以及何时实现警告和对话框。例如，您的应用程序可使用对话管理器来询问用户在退出之前是否要存储新的或修改过的文稿。如果没有足够的磁盘空间来存储文件，您的应用程序可借助对话管理器把这种情况告诉用户。

事实上所有的应用程序都要用到警告和对话框。为了避免不必要的开发工作，使用对话管理器来实现警告和创建大多数对话框。避开对话管理器而使用窗口管理器，控制管理器、QuickDraw和事件管理器例程来创建或响应在复杂对话框中的事件也是有可能的，甚至有时有这种需要。尽管您决定不使用对话管理器，阅读本章可帮助您了解关于有效的人机界面设计和对话框本地化问题的信息。

若要使用本章的内容，您应该熟悉资源、事件管理器、窗口管理器和控制管理器。

您一般使用资源来指定您希望在警告栏和对话框中显示的项目。例如，您在一个对话资源（'DLOG' 类型的资源）中指定一个对话框的大小、位置和外观。请参阅本书“Macintosh 工具箱引论”一章中有关资源的一般信息；在《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 的“资源管理器”一章中有关于资源管理器及其例程的详尽介绍。

对话管理器提供的例程处理大部分与警告和对话框相关的事件，但您的应用程序还需要处理几个额外的事件，6 - 86页的“为警告和定模式对话框编写事件过滤函数”一节中有关于这方面的介绍。请参阅本书“事件管理器”一章中有关事件和事件处理的一般信息。

对话管理器使用窗口管理器来显示您的警告和对话框。尽管对话管理器使用大部分必要的窗口管理器例程来激活和更新您的警告和对话框，但如果您的应用程序创建某些类型的对话框（如不定模式对话框，本章将介绍），您的应用程序还需要使用窗口管理器例程。请参阅本书“窗口管理器”一章中有关窗口的一般信息。

对话管理器使用控制管理器来创建和显示按钮、选项按钮、复选框和弹出式菜单，并处理它们中的事件。通常，您不应该在您的对话框中使用任何其它的控制部件（如滚动杆）。如果您需要实现一个更加复杂的控制部件，请参阅本书“控制管理器”一章。按钮是您可以在警告栏中使用的唯一的控制部件。

如果您在对话框中包括可编辑文字项目，对话管理器就使用 `TextEdit` 来处理相关的编辑任务。若想了解关于TextEdit的一般信息，请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 中的“`TextEdit`”一章。

本章简要地介绍了警告和对话框的概念和函数，然后讨论您怎样实现以下功能：

- 创建并显示警告和对话框
- 在您的警告和对话框中包括信息文字，可编辑文字域和其它类似的项目
- 响应您的警告和对话框中的事件

警告和对话框介绍

警告的行为和用途与对话框不同。同时在不同类型的警告之间，不同类型的对话框之间，均存在很大的区别。您可根据用户的当前状况在其中作出选择。

您的应用程序应该采用警告向用户报告错误或发出警告。

警告可以只向用户发出示警声，也可显示包含消息的警告栏并要求用户认可，或者两者均可采用。警告栏是包含信息文字、按钮，一般还有图标的特殊窗口。它们也可能包含图片。如图6 - 1所示，警告栏一般由说明为什么出现警告的文字和要求用户认可或纠正问题的按钮组成。

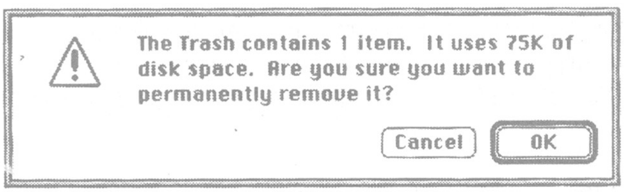


图6 - 1 Finder使用的一个警告栏

警告栏在继续工作之前强迫用户认可警告栏，要求用户单击一个按钮。为了帮助那些当警告栏出现时不知怎样响应的用户，您的应用程序为每个警告栏指定一个默认按钮，这个按钮调用一个默认动作，对话管理器围绕首选按钮绘制黑体轮廓线，使得默认按钮比警告栏中其它按钮突出。有轮廓线的按钮也就是警告栏的系统设定按钮；如果用户敲击 `Return` 键或 `Enter` 键，对话管理器就把这看成用户已经单击了默认按钮而作出相应动作。例如，如果用户在响应图6 - 1所示的警告栏时敲击 `Return` 或 `Enter` 键，对话管理器就反色显示“确定”按钮并通知Finder已经选定“确定”按钮；然后Finder作出响应，删除废纸篓中包含的项目。

当您的应用程序需要更多的信息来执行一个命令时，使用对话框。菜单中的命令一般仅作用于一个对象。如果用户选择的命令需要用户提供更多的信息您的应用程序才能执行它，就使用对话管理器从用户那里得到信息。如果一个命令需要产生一个对话框来获取更多信息，就在菜单中这个命令的名称之后放置省略号（…）向用户表明。

对话框是一种特殊的窗口，类似于一种表格，用户在其中核查各种项目并填充。图6－2给出了一个典型的对话框。

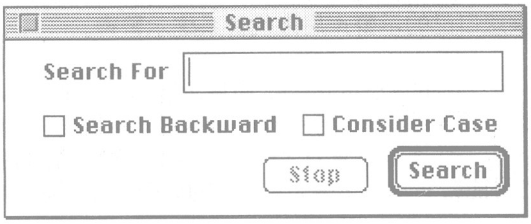


图6－2　一个典型的对话框

一个警告要继续进行一般只需得到用户认可，而一个对话框一般要求用户提供完成命令必要的信息(例如，通过输入文字或单击一个复选框)。当您创建执行一个命令的对话框时，您通常提供“确定”和“取消”按钮。当用户单击“确定”按钮时，您的应用程序应该根据用户在对话框中提供的信息执行命令。当用户单击“取消”按钮时，您的应用程序应该撤消命令并收回它所有的动作，就像用户从来没有发出这个命令一样。您可使用一个说明所执行的动作的按钮取代“确定”按钮。例如，您可在一个“搜索”命令的对话框中使用一个Search按钮，或在一个“取消”命令的对话框中使用一个Remove按钮。为简单起见，本章把执行对话框中所说明的动作的按钮作为“确定”按钮。您甚至可提供多个执行命令的按钮，每一种的方法略有不同。例如，在一个“更改”命令的对话框中，您可采用一个Change Selection按钮仅用来替换当前选定项，用Change All按钮来替换整个文稿中的所有发生的事件。

您可在您创建的对话框中使用下面的任意或全部元素：

- 信息或指导文字
- 可编辑文字区域
- 控制部件
- 图形（图标或QuickDraw图片）
- 您的应用程序定义的其它项目

警告类型

每个应用程序的每个用户都有可能做一些应用程序不能理解或不能以正常方式解决的事情。警告为您的应用程序提供一种以一致的方式响应这些情况的方法。警告的两个主要分类为：示警声和警告栏。

系统示警声是存储在系统文件中的语音资源。每当系统软件或您的应用程序使用语音管理器过程SysBeep时就发出这种语音。语音控制板允许用户选定哪种语音作为系统示警声。您也可以提供您自己的示警声用来代替系统示警声。

对于那些既小又能马上判别的明显的错误，使用示警声。例如，如果用户试图在简拼输入中采用全拼，您的应用程序就可发出示警声而不用显示警告栏。您的应用程序可以使它的响应基于一个警告条件连续重复发生的次数。第一次时，您的应用程序可能只发出示警声，而这以后它可能会展示一个警告栏。您的应用程序可以为4个警告阶段定义不同的响应。

警告栏主要是从您的应用程序到用户的单向通讯；用户响应的唯一方法是单击“确定”或“取消”按钮。因而您的警告栏应该包含按钮，但它们不应该包含可编辑文字域、选项按钮或复选框（这些项目一般在对话框中出现）。

有三种标准类型的警告栏：“注意”警告栏、“当心”警告栏和“停止”警告栏。它们通过它们左上角的图标相互区别。

如果对某种情况即使不加过问也不会带来任何灾难性的后果，就使用“**注意**”警告通知用户。一般说来这种类型的警告只提供信息，用户通过点按“确定”按钮来响应它。如图6－3所示，有时一个“注意”警告也会提出简单的问题并提供可供选择的响应。

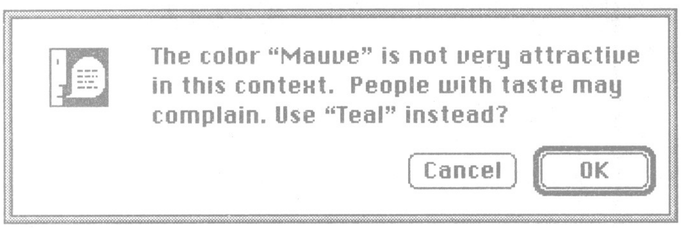


图6－3 “注意”警告

如果允许一个操作继续下去会产生不满足需要的结果，就使用“当心”警告向用户发出警告。如图6 – 4所示，您应该让用户选择是继续动作（单击“确定”按钮）还是停止动作（单击“取消”按钮）。

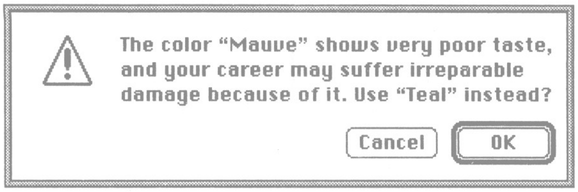


图6 – 4“当心”警告

如果一个问题或情况太严重以致于动作不能完成，就使用“停止”警告来通知用户。如图6 – 5所示，“停止”警告一般只有一个按钮（“确定”），因为用户所能做的只是认可该动作不能完成。

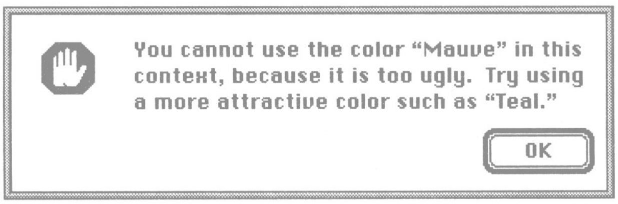


图6 – 5 “停止”警告

您也可创建自定义警告栏，在其左上角包含您自己的图标和空格。本书前面的版面2给出了一个警告栏，当用户在苹果菜单中选择“关于”命令时，Surfwriter应用程序显示这个警告栏。用户在阅读这个警告栏中的信息后，单击“确定”按钮消除它。

对话框的类型

对话框应该总是要求从用户那里获取信息，同时也把信息传达给用户。这就是说，对话框的目的是在用户和您的应用程序之间进行对话，这种通讯一般是为执行一个命令作准备。您的对话框可包括可编辑文字域和控制部件（如复选框和选项按钮）。有了这些，用户就可为您的应用程序提供执行命令所需的信息。有三种对话框类型：定模式对话框、可移动定模式对话框和不定模式对话框。以下3部分中将对这些类型加以介绍。

定模式对话框

在允许用户继续进行任何其它工作之前，许多对话框要求用户点按一个按钮。如果当用户在对话框之外单击时用户接收到的唯一响应就是示警声。这种类型的对话框叫做定模式对话框，因为它将用户置于这种状态或“方式”：只在对话框中才能工作。定模式对话框也叫做固定位置定模式对话框（以便它区别于可移动定模式对方框），这种类型的对话框看起来像除包括按钮外还包括其中类型控制部件的警告栏。图6－6给出一个定模式对话框，用户选择“拼写检查”命令后Surfwriter显示这个对话框。

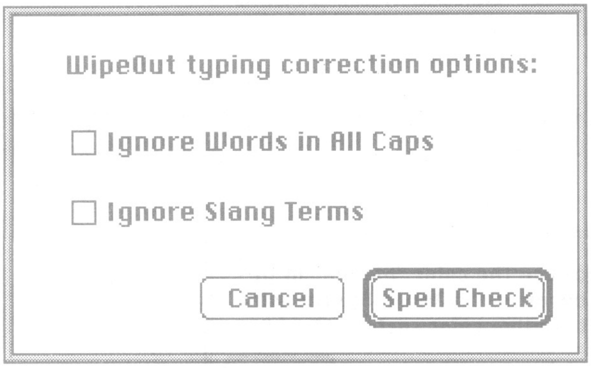


图6－6 定模式对话框

重要告诫

由于用户在做任何其它事情之前必须显式地消除定模式对话框，所以只有当用户必须在执行任何其它工作之前完成一个操作时您才应使用定模式对话框。固定位置定模式对话框限制了用户的行动自由，因而要有节制地使用它们。一条首要原则是，只要有可能就使用不定模式对话框，尽量多使用可移动定模式对话框，只有在您无法实现不定模式可移动对话框时才使用固定位置定模式对话框。

一个定模式对话框通常至少有两个按钮：“确定”和“取消”。当用户单击“确定”按钮时，您的应用程序应该根据用户提供的信息执行命令并取消定模式对话框。如果您愿意，您可给“确定”按钮取一个更切题的标题。当用户单击“取消”按钮时，您的应用程序应该取消自显示定模式对话框以来它所做的所有动作，然后取消定模式对话框。一定要用“取消”作为这个按钮的标签。您的对话框也可以有额外的按钮，这些按钮也可能消除对话框。

您创建的每个对话框都应该有一个系统设定按钮。这就是说，当用户敲 Return 或 Enter 键时，就调用这个系统设定按钮的动作。如果您不提供您自己的事件过滤函数，对话管理器就把您在对话框说明中指定的第一个项目作为系统设定按钮(只要第一个项目是一个按钮)。您可以使用事件过滤函数来完善对话管理器处理事件的能力，在6 - 86页的"为警告和定模式对话框编写事件过滤函数"一节中有关于事件过滤函数的介绍。例如，一个事件过滤函数还能检测磁盘插入事件，允许后台应用程序接收更新事件。如果您提供您自己的事件过滤函数，它应该检测涉及 Return 和 Enter 键的键按下事件，就像系统设定按钮被选定一样作出响应。系统设定按钮应该调用默认动作，并且您应该尽量完善设计以保证默认动作的安全，使它不会丢失数据。

尽管对话管理器围绕警告栏中的系统设定按钮绘制黑体的轮廓线，但它并不给对话框中的系统设定按钮绘制黑体轮廓线。若要表明默认动作，您的应用程序应该绘制系统设定按钮的轮廓线。6 - 56页的"使用定义应用程序的项目绘制按钮的黑体轮廓线"部分给出了一种方法，您可用这种方法给按钮绘制轮廓线。如果您不给对话框中的按钮绘制轮廓线，那么就没有系统设定按钮，并且您必须在您的事件过滤函数中保证按下 Return 或 Enter 键不会产生任何作用。

可移动定模式对话框

用户有时需要察看被一个叠加的定模式对话框遮住的窗口。在这种情况下，您应该使用一个可移动定模式对话框，不要用固定位置定模式对话框。可移动定模式对话框是具有标题栏的定模式对话框，用户可通过拖拽标题栏移动对话框。

可移动定模式对话框不包含关闭格，也不应包含缩放格。但具有标题栏，这表明用户可移动对话框，但对话框仍然是定模式的，即用户必须在执行您的应用程序中的任何其它工作之前响应对话框。如果用户单击属于您的应用程序的另一个窗口，它就应该发出系统示警声。只有在用户单击可移动定模式对话框的一个按钮之后，您的应用程序才取消此对话框。但是，这种类型的对话框不像正规的定模式对话框，它允许用户通过单击应用程序的一个窗口、或者从应用程序菜单或苹果菜单中选择应用程序的名称，把另一个应用程序切换到前台来。

图6 - 7给出了一个可移动定模式对话框。在用户从文件菜单中选择 Find 命令之后，Finder 就显示这个对话框。

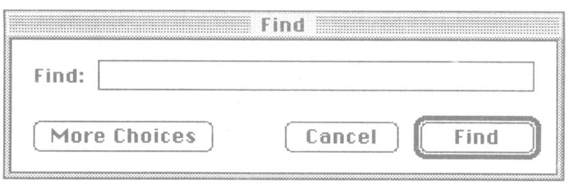


图6 - 7 可移动定模式对话框

很有必要考虑是否可用不定模式对话框取代定模式对话框或可移动定模式对话框，这样特别有利于保护用户以任何顺序执行任何任务的自由。

可移动定模式对话框一般应该像定模式对话框一样响应。注意，当您显示一个可移动定模式对话框时用户应该能够在您的应用程序和另一个应用程序之间切换(这样就把您的应用程序切换到后台)，而在定模式对话框中用户却不能执行这个动作。例如，Macintosh系统软件可使用几个可移动定模式对话框来表明Finder正忙于一个耗时操作（如文件拷贝)，但用户仍可把 Finder 切换到后台。

不定模式对话框

另外的对话框并不要求用户在做任何其它事情之前作出响应，这些对话框叫做不定模式对话框。只要有可能，您就应该尽量采用不定模式对话框。如图6－8所示，不定模式对话框看起来像一个文稿窗口。用户能够移动它，激活或失活它，并像对待任何文稿窗口一样关闭它。与文稿窗口不同是，它主要由按钮和其它控制部件组成，而不是文字，并且它不包含滚动杆和尺寸格。(不定模式对话框不应该有尺寸格或滚动杆；如果您需要这些控制部件，使用窗口管理器来创建一个窗口。)

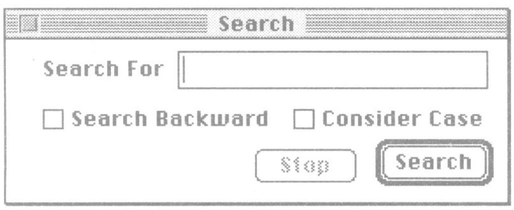


图6－8 不定模式对话框

当您显示一个不定模式对话框时，您必须允许用户无需消除对话框就能执行其它操作（如在文稿窗口中工作)。当用户在不定模式对话框中单击一个按钮时，您的应用程序不应该取消它；相反，对话框应该保留在桌面环境中使用户可以再执行它。由于很难取消不定模式对话框最后调用的动作，所以它一般没有“取消”按钮，但它可以有一个 Stop 按钮。例如，要终止长时间的打印和搜索工作，不定模式对话框中的 Stop 按钮就很有用。

当完成不定模式对话框的工作后，用户可以单击它的关闭格或从文件菜单中选择关闭（当对话框是当前活动窗口时)。然后您的应用程序就应该取消不定模式对话框。当用户选择退出时，不定模式对话框也被隐式地消除。您的应用程序在对话框消除之后记住它的内容对用户有用。这样的话，当用户再调用对话框时，甚至在用户关闭和重新打开您的应用程序之后，您可以把对话框还原成与以前的一模一样。

警告和对话框中的项目

所有的对话框和警告栏都包含项目——如图标、文字、控制部件和QuickDraw图片等。您可以用项目列表的资源来指定您要让哪些项目在您的警告栏和对话框中出现。您甚至可定义您自己的项目（例如，外观可更改的图片）。图6－9图解说明了大多数项目类型。

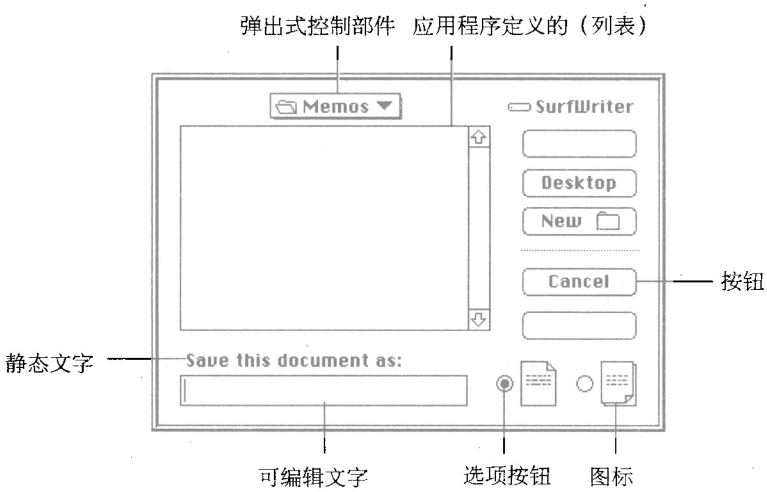


图6－9 对话框中的典型项目

您的应用程序使包括在其对话和警告栏中的项目有效或无效。对于有效项目，对话管理器报告涉及该项目的用户事件。例如，当用户单击图6－9中有效的“取消”按钮时，对话管理器会向应用程序报告。无效项目即对话管理器不报告其用户事件的项目。例如，当用户单击或拖拽图6－9中的静态文本项目“存储这个文稿为”时，对话管理器并不向应用程序报告，因为这个项目是无效的。

不要把无效项目和无效控制部件混淆。当您不想让控制管理器显示控制部件中的鼠标事件的可视响应时，您通过使用控制管理器过程HiliteControl使它变成无效。例如，直到用户输入一个文件名称，图6－9中的Save按钮还是无效。控制管理器用某种方式（如使之变暗）显示一个无效控制部件以表示它是不使能的。对话管理器并没使有效项目和无效项目之间存在任何视觉区别。唯一的区别是当用户单击一个无效项目时对话管理器不通知您的应用程序。

每当用户不能使用对话框中的控制部件时，您应该使用 HiliteControl 使这个控制部件彩色变暗淡。例如，图 6 - 8 给出了 Stop 按钮彩色变淡的不定模式对话框。Stop按钮变暗是因为只有用户单击Search按钮它才起作用。当用户通过单击Search按钮启动搜索操作时，Stop 按钮变成现用，而 Search 按钮则变暗。

当您停用一个不定模式对话框或可移动定模式对话框时，您应该使用控制管理器过程 HiliteControl 使其中的按钮和其它控制部件成为不使能。HiliteControl 过程用暗淡的彩色显示无效的按钮、选项按钮、复选框和弹出式菜单，向用户表明当对话框在后台时单击这些项目不起任何作用。当您再激活对话框时，使用 HiliteControl 把这些控制部件重新变成有效。

您在一个项目列表资源中储存关于所有对话或警告栏项目的信息。当您使用对话管理器例程来调用警告栏或创建对话框时，对话管理器从资源中获取关于它们的大部分说明信息。对话管理器调用资源管理器把它所需的信息从资源中读入存储器。

警告和对话框中的事件

处理警告栏中的事件很简单：当您调用警告栏之后，对话管理器自动地调用 ModalDialog 过程为您处理大部分事件。

若要处理定模式对话框中的事件，在显示对话框之后，您的应用程序必须显式地调用 ModalDialog 过程。

在这两种情况中，当一个有效的项目被单击时，对话管理器均返回该项目的号码。然后您将作出响应完成适当的工作。对于在警告栏或定模式对话框之外的鼠标按下事件，ModalDialog 过程发出系统示警声并获取下一个事件。

当用户单击任何有效的项目时，对话管理器自动地取消警告栏。而对于定模式对话框来说，您的应用程序应该继续调用ModalDialog过程直到用户选择确定或取消按钮为止，并且在对用户的选定项作出相应的响应之后，您的应用程序应该取消对话框。

当 ModalDialog 接收到一个事件时，ModalDialog 在它自己处理事件之前把这个事件传递给一个事件过滤函数。您应该为ModalDialog不处理的事件提供一个事件过滤函数作为一个辅助事件处理循环。对于警告和定模式对话框，您应该提供一个执行下列任务的简单的事件过滤函数：

- 如果用户敲击 Return 或 Enter 键，返回 TRUE 和项目的号码
- 如果用户敲击 Esc 键或“Command” + “·”组合键，返回 TRUE 和“取消”按钮的项目号码
- 响应更新事件更新您的窗口（这也允许后台应用程序接收更新事件）并返回 FALSE
- 对于您的事件过滤函数不能处理的所有事件，返回 FALSE

您也可使用事件过滤函数来检测并响应等效命令组合键和更复杂的事件（例如，用户在一个定义应用程序的项目内拖拽光标）。

对于您的应用程序的不定模式或可移动定模式对话框，您可把事件传递给 IsDialogEvent 函数，或者用您自己的事件处理代码判别事件是否需要当作对话框的部分来处理。如果需要，调用 DialogSelect 函数来处理它们，不要调用 ModalDialog 过程。当不定模式对话框是当前活动窗口时，您的应用程序不应该取消不定模式对话框除非用户单击它的关闭格或在文件菜单中选择关闭。只有在用户单击可移动定模式对话框的一个有效按钮后，您的应用程序才可以取消这个可移动定模式对话框。

除调用 IsDiaogEvent 或 DialogSelect 函数来处理不定模式和可移动定模式对话框中的事件外，您还可不必借助对话管理器的帮助，使用控制管理器、窗口管理器和 TextEdit 例程（如 FindWindow，BeginUpdate，EndUpdate，FindControl，TrackControl 和 TEClick 来处理这些事件。

警告栏、对话框和窗口管理器

对话管理器使用窗口管理器来绘制您的警告栏和对话框。就像对待任何其它窗口一样，您可使用窗口管理器或 QuickDraw 例程来操纵警告栏和对话框（显示、隐藏、移动它并调整其大小）。

对话管理器从一个资源文件的资源中获取关于警告和对话框的大部分说明信息。警告资源是说明警告的资源，对话资源是说明对话框的资源。这两种资源均与窗口资源类似。（除了提供对话管理器传递给窗口管理器的信息以外，您还在您的警告资源和对话资源中包括对话管理器单独使用的额外的信息。6 – 18 页的“创建示警声和警告栏”部分和 6 – 23 页的“创建对话框”部分对这些资源作了全面的介绍。）

当您创建警告栏时，对话管理器总是将警告栏的 dBoxProc 窗口定义标识符传递给窗口管理器；这就是为什么所有警告栏有同样的标准外观和行为的原因。窗口管理器总是在所有其它窗口前面显示警告栏。因为警告栏要求用户在做另外的任何事情之前作出响应，并且所作的响应会消除警告栏，所以您的应用程序一般不需要使用任何窗口管理器或 QuickDraw 例程来操纵警告栏。

创建对话框的函数 GetNewDialog 与窗口管理器函数 GetNewWindow 相似。当您创建一个对话框时，您提供的信息与您用 GetNewWindow 创建窗口时提供的信息一样。例如，您使用一个资源指定窗口定义标识符（它确定对话框的外观和行为方式）和定义对话框图形端口尺寸的矩形。像对任何窗口一样，您指定对话框的放置位置（按惯例最初应该处在最前面），还指定它最初是可视还是不可视。例如，如果您创建一个最初不可视的对话框，如果您在显示它之前需要设定一个控制部件的值，您就使用窗口管理器过程 ShowWindow 来显示对话框。

对话管理器通过调用窗口管理器函数NewCWindow来创建对话窗口，然后在窗口记录中设定窗口类型，表明它是一个对话框。用来处置对话框的对话管理器过程CloseDialog和DisposeDialog与窗口管理器过程CloseWindow和DisposeWindow相似。

当您创建对话框时（6 – 23页的“创建对话框”一节中有相关的介绍），针对定模式对话框使用dBoxProc窗口定义标识符。针对不定模式对话框使用noGrowDocProc窗口定义标识符。（如果您的对话框确实需要尺寸格和滚动杆，您应该使用窗口管理器来创建窗口，不要使用对话管理器。）最后，使用movableDBoxProc窗口定义标识符来创建可移动定模式对话框。

对话管理器提供处理警告栏和对话框中大部分事件的例程。例如，您的应用程序并不需要使用像窗口管理器函数FindWindow和控制管理器函数TrackControl这样的例程来确定警告栏的按钮内鼠标按下事件发生的时间和地点，您的应用程序只需要相应地响应鼠标单击。对话管理器也自动地为您的警告栏和对话框处理更新和激活事件。6 – 77页的“处理警告栏和对话框中的事件”一节详细地介绍了怎样使用对话管理器来帮助您的应用程序处理事件。

关于对话管理器

对话管理器极大地简化了创建警告栏和简单定模式对话框的工作。每当您需要创建警告栏时，借助对话管理器将使您事半功倍。（如果在出现错误时您只想发出系统示警声而不显示警告栏，您可使用语言管理器过程SysBeep替代对话管理器。若想了解关于SysBeep过程更多的信息。请参阅《Inside Macintosh 声音》*(Inside Macintosh : Sound)* 一书。

但是，如果您要创建很复杂的对话框，您就会发现使用对话管理器的益处开始减弱。对于复杂的定模式对话框（特别是那些包含多部分控制部件或多个定义应用程序的项目的定模式对话框）和许多可移动定模式和不定模式对话框，使用窗口管理器来创建标准窗口以实现您自己的对话框，使用控制管理器、QuickDraw和事件管理器来处理对话管理器承担的任务，这样做会更方便一些。

在决定是否使用对话管理器时要考虑两个主要问题：

- 是否使用窗口管理器和控制管理器取代对话管理器来创建对话框

■ 是否使用事件管理器、窗口管理器、控制管理器，以及 TextEdit 代替对话管理器来处理事件

例如，您可能希望使用对话管理器来创建复杂的对话框，然后使用事件管理器、窗口管理器、控制管理器和 TextEdit 来处理您的正常事件循环内的事件。对于可移动定模式对话框和不定模式对话框，在本章所列出的示例代码中使用这样一种混合方法：使用对话管理器来创建对话框，但使用一般的事件处理代码，以便根据最前面的窗口类型来确定相应的动作。当一个不定模式对话框或可移动定模式对话框位于前面时，本章举例说明了针对该对话框如何进行动作。

如果您不使用对话管理器在标准窗口中绘制您自己的对话框，您就不能使用对话管理器的例程来帮助处理事件，但另一方面，您可以更快地更新窗口，并且更方便地扩展其事件处理功能。在下面这些情况中，使用对话管理器来创建对话框或处理涉及它们的事件就不甚方便：

■ 对话框中包含 20 个以上的项目
■ 您需要一项由多个部分组成的控制部件，如滚动杆
■ 您需要将项目移动到屏幕上或移出屏幕
■ 您需要显示一个可移动的指示符，如一个进度指示符
■ 您需要在对话框中显示一个列表［关于列表的详细说明，请参阅《Inside Macintosh 工具箱扩展》(*Inside Macintosh: More Macintosh Toolbox*) 的“列表管理器”一章。］
■ 您需要用非系统字体显示文字
■ 当您的应用程序显示对话框时，您的应用程序必须对某些事件作出响应，但这些事件不包括鼠标按下事件、在可编辑文字项目内的键按下事件以及用于等效命令组合键的几个键按下事件

如果您希望创建的对话框不会有上述这些情况，那么，您当然应该使用对话管理器。如果只有其中一种情况，您仍可以使用对话管理器。如果有两种以上的情况出现，您会发现，与使用对话管理器来创建或管理对话框相比，创建并管理可像对话框一样操作的标准窗口则更为适宜。

使用对话管理器

您可以使用对话管理器完成：

■ 警告用户出现了紧急情况
■ 当您的应用程序需要用户进行输入信息时，与用户进行对话

您可以利用对话管理器的例程来请求窗口中的警告栏或创建对话框，而警告栏和对话框的内容又是由对话管理器来管理的。对话管理器自动地为您的警告栏和对话框处理更新事件、激活事件、光标跟踪，以及大多数文字编辑任务。

若要执行警告栏和对话框，您通常应

- 在资源文件中创建警告资源或对话资源
- 创建另一个资源来指定项目列表——如控制部件、信息文字，以及图片等可在警告栏或对话框中显示的项目
- 创建并显示警告栏或对话框
- 对涉及您的警告栏或对话框的事件作出相应的响应
- 当您结束对话时，关闭对话框（对于警告栏，对话管理器自动为您执行关闭动作）

本章随后将详细解释这些任务。

在使用对话管理器以前，您必须依次初始化QuickDraw、字体管理器、窗口管理器、菜单管理器和TextEdit。然后，使用InitDialogs过程来初始化对话管理器。

对话管理器使用系统示警声在各个警告阶段向用户发出信号。如果您希望使用其它示警声，而不是系统示警声，那么请编写您自己的语音过程（如程序清单6－3所示），并且调用ErrorSound过程使它成为当前的示警声过程。

如果您希望用某种非系统字体显示静态文字或可编辑文字，您可以使用SetDialogFont过程。但是，关于此过程有一些防止误解的说明。请参阅6－105页的“特殊考虑”中的SetDialogFont一段关于这些防止误解的说明。

System 7和早期版本的通讯工具箱添加了若干例程（即AppendDITL、ShortenDITL和CountDITL），使您更易在对话框中添加项目、删除项目和计算项目数量。在调用这些例程以前，您应该使用具有gestaltDITLExtAttr选择程序的Gestalt函数来确保它们是有效的。测试一下由响应参数中gestaltDITLExtPresent常数所表示的位域。如果该位是设定的，那么，AppendDITL，ShortenDITL和CountDITL是有效的。

```
CONST gestDITLExtPresent = 0;   {if this bit is set,then}
                                {AppendDITL,ShortenDITL,}
                                {& CountDITL are available}
```

在《Inside Macintosh 操作系统实用程序》(*Inside Macintosh: Operating System Utilities*) 的“格式塔管理器”一章中介绍了Gestalt 函数。

创建示警声和警告栏

若要创建一个警告，就可以使用这些函数中的一个：NoteAlert，CautionAlert，StopAlert和Alert。如前面的图6－3、图6－4和图6－5所示，与前三个函数相关的图标显示在警告栏的左上角。Alert函数允许您在警告栏的左上角显示您自己的图标或者根本不显示图标。这些函数可以从您提供的警告资源中得到关于警告的说明性信息。警告资源的资源类型是‘ALRT’。当您调用其中一个函数时，应把警告资源的资源标识符和指向一个事件过滤函数的指针传送给它。这些函数可以创建并显示警告栏。当用户在一个警告栏中单击一个按钮时，这些函数便返回该按钮的项目编号并关闭警告栏，此时您应对用户的单击作出相应的响应，6－81页的“响应警告栏中的事件”对此作了介绍。

下面是一个创建图6 - 10所示的“当心”警告栏的示例。

```
VAR
    myAlertItem:            Integer;
myAlertItem := CautionAlert(kSaveAlertID, @MyEventFilter);
```

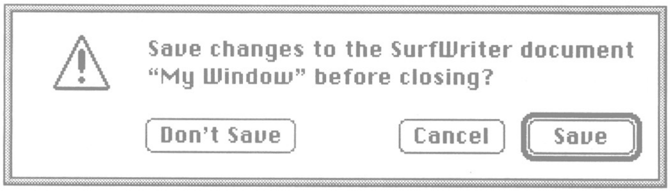


图6 - 10　告诫用户存储更改过的文稿的警告栏

当您调用Alert，StopAlert，CautionAlert和NoteAlert函数时，您应该指定一个指向一个事件过滤函数的指针。您还应提供一个事件过滤函数作为一个次事件处理循环，以处理ModalDialog不能处理的事件。在此实例中，一个指向MyEventFilter的指针被指定用于事件过滤函数。通过在此参数中传送NIL，您就可以使用标准事件过滤函数。标准事件过滤允许用户按Return或Enter键来代替单击系统设定按钮。根据6 - 86页的“为警告栏和定模式对话框编写事件过滤函数”中的介绍，您的应用程序应该提供一个简单的事件过滤函数，该函数也允许后台应用程序接收更新事件。您可以在大多数或所有的警告栏和定模式对话框中使用相同的事件过滤函数。

接着前面的实例，由应用程序定义的常数（kSaveAlertID）在一个参数中指定用于CautionAlert函数警告资源的资源标识符。程序清单6 - 1表示此警告资源是如何以Rez输入格式出现的。（Rez是苹果公司的Macintosh程序员工作间MPW提供的资源编译程序，从APDA那里可以购得。）

程序清单6 - 1 一个警告资源的Rez输入

```
resource 'ALRT' (kSaveAlertID, purgeable) {  /*alert resource*/
    {94, 80, 183, 438},     /*rectangle for alert box*/
    kSaveAlertDITL,         /*use the 'DITL' with res ID 200*/
    {                       /*alert stages, starting with #4; at each */
                            /* stage, make OK the default, display the */
                            /* alert box, & play the system alert sound*/
    OK, visible, sound1,    /*4th consecutive error*/
    OK, visible, sound1,    /*3rd consecutive error*/
    OK, visible, sound1,    /*2nd consecutive error*/
    OK, visible, sound1,    /*1st error*/
    },
    alertPositionParentWindow  /*place over document window*/
};
```

一个警告资源中包含下列信息:

- 在全局坐标中给定的一个矩形，它可以确定警告栏的尺寸，或者它的位置（可任选）；这些坐标指定了警告栏的左上角和右下角
- 该警告栏的项目列表的资源标识符
- 在四个警告阶段中的各阶段要采取的动作
- 作为选项的一个常数（alertPositionParentWindow, alertPositionMainScreen 或者 AlertPositionParentWindowScreen）用于通知对话管理器将警告栏定位在哪里（仅适用于在 System 7 中运行的应用程序）

在程序清单6－1中，坐标（94，80，183，438）指定了警告栏的尺寸，alertPositionParentWindow 常数使得对话管理器将警告栏放置于紧靠用户文稿窗口的标题栏下面。如果您没有提供定位常数，对话管理器便将警告栏放置在您为警告栏的矩形指定的全局坐标中。6－62页的“定位警告和对话框”中介绍了用于警告栏的定位常数。

在程序清单6－1中，应用程序定义的常数 kSaveAlertDITL 代表项目列表资源的资源标识符。6－26页的“为警告和对话框提供项目”介绍了如何创建一个项目列表资源。

您的应用程序可以根据重复出现警告条件的相继次数予以响应。在程序清单6－1中，警告资源指定了用户重复动作（该动作请求此“当心”警告栏）的各个相继时间，对话管理应该执行如下的动作：将“确定”按钮加上轮廓线，并且视它为系统设定按钮，显示警告栏（即把它变成“可视的”），并且发出一次系统示警声。

您的应用程序可以为四个警告阶段中的每个阶段定义不同的响应。这样最适于停止某些警告信号——如那些表示动作不能完成的警告，尤其是极有可能发生的意外动作（例如，当没有选定剪贴文字时用户就选择 Cut 命令）。在这种情况下，您的应用程序可能在用户前两次操作有误时仅发出系统示警声，对于随后的操作错误仍可能显示出一个警告栏。甚至在第四个警告阶段以后相继发生的错误也被视为第四阶段警告。

例如，用户也许会尝试将一个图形贴到一个简单页面布局程序的页面边缘之外；用户第一次这样尝试时，该应用程序（使用对话管理器）可能只是向用户发出系统示警声。如果用户重复犯这样的错误，应用程序可能再发出系统示警声。但当用户连续第三次重复这一错误时，应用程序可能显示一个如图6－11所示的警告栏。如果用户在消除了此警告栏以后立即犯了同样的错误，则该警告栏再次出现，并且连续如此，直到用户纠正错误或者放弃不正确的动作为止。

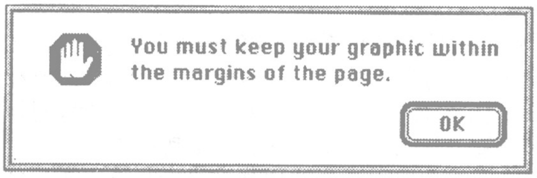


图6－11　仅在第三和第四警告阶段显示的一个警告栏

在程序清单6－2中所示的警告资源用于指定图6－11所示的“停止”警告信息。请注意，第四警告阶段列在第一，而第一警告阶段列在最后。在第三警告阶段，
显示一个警告栏，但不发出系统示警声。如果用户连续第四次犯同样错误，应用程序发出系统示警声，并且再次显示警告栏。

程序清单6－2 根据警告阶段来指定不同的警告响应

```
resource 'ALRT' (kStopAlertID, purgeable) {  /*alert resource*/
    {40, 40, 127, 353},      /*rectangle for alert box*/
    kStopAlertDITL,          /*use the 'DITL' with res ID 300*/
    {                        /*alert stages, starting with #4*/
    OK, visible, sound1,     /*4th err: show alert box, play alert sound*/
    OK, visible, silent,     /*3rd err: show alert box, don't play sound*/
    OK, invisible, sound1,  /*2nd err: play sound, don't show alert box*/
    OK, invisible, sound1,  /*1st err: play sound, don't show alert box*/
    },
    alertPositionParentWindow  /*place over document window*/
};
```

下面的三项说明用来指定每个警告阶段的动作：

- 哪个按钮是系统设定按钮——“确定”按钮（即项目列表资源中的第一个项目）或“取消”按钮（即项目列表资源中的第二个项目）对话管理器自动地在系统设定按钮的周围绘制一个黑体轮廓线，并且当用户按下Return或Enter键时，对话管理器或您的事件过滤函数应该视此键盘事件为单击系统设定按钮。6－26页的“为警告栏和对话框的提供项目”中介绍了“确定”按钮和“取消”按钮。您可以在每个警告阶段更改系统设定按钮，尽管这种更改系统设定按钮的方案对用户是否有帮助很难定论。在前面的实例中，系统设定按钮是“确定”按钮。

- 是否需要显示警告栏。如果您为一个警告阶段指定了 visible 常数，则显示出警告栏；如果您指定了 invisible 常数，则不显示出警告栏。在程序清单 6－2 中，当用户前两次连续重复错误时，不显示出警告栏，但若随后连续发生错误，则显示出警告栏

- 在此警告阶段应该发出四种可能的语音中的哪一种(如果有的话)。在前面的实例中，第一、第二和第四警告阶段发出一次系统示警声，但在第三阶段不发出语音

按照系统设定值，对话管理器使用系统示警声。在程序清单6－2中使用的sound1常数通知对话管理器发出一次系统示警声；您也可以指定 sound2 和 sound3 常数，使对话管理器发出两次或三次系统示警声，每一次的间隔和持续时间相同。语音的音量取决于当前扬声器音量的设定状态，用户可以在语音控制板中调整音量。如果用户将扬声器的音量设定成0，则菜单栏用一次闪烁来取代其它设定状态下用户能听到的每个语音。

如果您希望对话管理器不使用系统示警声来发出示警声，您可以编写自己的语音过程，然后调用 ErrorSound，并且向它传送一个指向您的语音过程的指针。ErrorSound 过程（在第 6－104 页上介绍）可以使您的语音过程成为当前语音过程。例如，您可以创建一个叫作 MyAlertSound 的语音过程，如程序清单 6－3 所示。

程序清单 6－3 创建您自己的示警声过程

```
PROCEDURE MyAlertSound (soundNo: Integer);
BEGIN
     CASE soundNo OF
     0:   PlayMyWhisperAlert;{sound for silent constant in alert resources}
     1:   PlayMyBellAlert;   {sound for sound1 constant in alert resources}
     2:   PlayMyDrumAlert;   {sound for sound2 constant in alert resources}
     3:   PlayMyTrumpetAlert;{sound for sound3 constant in alert resources}
     OTHERWISE                    ;
     END;                         {of CASE}
END;
```

对于SoundNo参数中报告的四个警告阶段的每个阶段，您的过程可以发出您定义的任何示警声。

如前面所解释的，您在警告资源中指定矩形尺寸可以确定警告栏的尺寸。您还可以使该矩形坐标作为定位警告栏的全局坐标，或者您可以让对话管理器根据三个标准位置来自动地定位警告栏。例如在 6 － 21 页的程序清单 6 － 2 中，使用常数 alertPositionParentWindow将警告栏定位在用户正在工作的文稿窗口上。关于这些标准位置的详细说明，请参阅 6 － 62 页的“定位警告栏和对话框”。

创建对话框

若要创建对话框，可以使用 GetNewDialog 或者 NewDialog 函数来实现。通常您应该使用 GetNewDialog，它是可从资源文件中一个对话（‘DLOG’）资源中获取有关对话框的信息。像窗口资源一样，对话资源与您应用程序的说明性信息是分离的，以便易于修改或译成其它语言。本节的其余部分介绍了如何使用GetNewDialog。虽然一般不推荐您使用NewDialog或者NewColorDialog函数，但您还是可以使用它们，并且在各个参数中向它们传送必需的说明性信息，而不必使用对话资源。请参阅6 － 118页中关于 NewDialog 的说明以及 6 － 115 页中关于 NewColorDialog 的说明。

GetNewDialog利用对话资源中的信息来创建对话记录类型的数据结构，并且向它返回一个指针。一个对话记录中包含一个窗口记录。当您使用GetNewDialog时，对话管理器在窗口记录中将 WindowKind 域设定成 dialogkind。如6 － 61 页的“显示警告栏和对话框”中所解释的，您可以使用此指针和窗口管理器或 QuickDraw 的例程来显示并操纵对话框。

当您使用GetNewDialog时，您应向它传送对话资源的资源标识符，一个是用于对话记录的存储器的可选指针，另一个是窗口指针 Pointer（－1），该指针可以使窗口管理器在所有其它窗口的前面显示对话框。

如果您传送的存储器指针是NIL，对话框则被分配到您的应用程序堆中。对于定模式对话框和可移动定模式对话框，适合传送 NIL，但是，如果您创建的是一个不定模式对话框，这样会导致您的应用程序堆发生堆破碎。因此，对于不定模式对话框，您应该像处理窗口一样分配您自己的内存；在本书的“窗口管理器”一节中介绍了如何分配窗口存储器。

下面是一个实例，说明如何创建如图 6 － 12 所示的对话框。

```
VAR
    theDialog:              DialogPtr;
theDialog := GetNewDialog(kSpellCheckID, NIL, Pointer(-1));
```

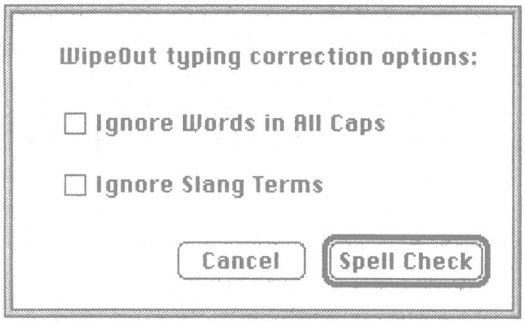


图 6 - 12　一个简单的定模式对话框

此实例使用一个应用程序提供的常数(kSpellCheckID)指定一个对话资源的资源标识符编号。程序清单 6 - 4 表示此对话资源是如何以 Rez 输入格式出现的。

程序清单 6 - 4 一个对话资源的 Rez 输入

```
resource 'DLOG' (kSpellCheckID, purgeable) { /*dialog resource*/
    {62, 184, 216, 448},    /*rectangle for dialog box*/
    dBoxProc,               /*window definition ID for modal dialog box*/
    visible,                /*display this dialog box immediately*/
    noGoAway,               /*don't draw a close box*/
    0x0,                    /*initial refCon value of 0*/
    kSpellCheckDITL,        /*use item list with res ID 400*/
    "Spellcheck Options",   /*title if this were a modeless dialog box*/
    alertPositionParentWindow   /*place over document window*/
};
```

对话资源中包含如下信息:

- 在全局坐标中给定的一个矩形，它可以确定对话框的尺寸和位置（可选）；这些坐标指定了对话框的左上角和右下角
- 窗口定义标识符，它可以指定窗口定义函数和对话框类型的变量代码
- 一个常数 (visibleªÚinvisible)，它可以指定是否立即在屏幕上绘制对话框
- 一个常数 (noGoAwayªÚgoAway)；goAway 仅用于指定不定模式对话框的标题栏中的关闭格
- LongInt 类型的参考值，您的应用程序可能把它用于各种目的
- 用于对话框的项目列表资源的资源标识符

- 一个文字字符串，用于不定模式对话框或者可移动定模式对话框的标题
- 一个作为选项的常数（alertPositionParentWindow、alertPositionMainScreen或AlertPositionParentWindowScreen），它可以通知对话管理器如何定位对话框（仅适用于在System 7上运行的应用程序）

在此实例中，坐标是（62，184，216，448）的矩形指定了对话框的尺寸，并且常数alertPositionParentWindow可以使对话管理器将对话框放置在用户文稿窗口的标题栏正下方。如果您没有提供一个定位常数，对话管理器便将对话框放置在您为对话框矩形所指定的全局坐标上。6－62页的“定位警告栏和对话框”介绍了用于对话框的定位常数。

在此实例中，使用了窗口定义标识符BoxProc。您可以使用下列窗口定义标识符来指定对话框的类型：

窗口定义标识符	**对话框类型**
dBoxProc	定模式对话框
movableDboxProc	可移动定模式对话框
noGrowDocProc	不定模式对话框

对于各种类型的对话框，对话管理器都使用窗口管理器来绘制相应的窗口框架。图6－6所示的定模式对话框实例是以dBoxProc窗口定义标识符来绘制的，图6－7所示的可移动定模式对话框实例是以movableDBoxProc窗口定义标识符来绘制的，图6－8所示的不定模式对话框是以noGrowDocProc定义标识符来绘制的。

程序清单6－4指定了visible常数，因此可以立即绘制出对话框。如果您使用invisible常数，则只有您的应用程序使用窗口管理器的过程ShowWindow来显示该对话框后，才被对话框绘制出来。

只有不定模式对话框才可以使用goAway常数。对于定模式对话框和可移动定模式对话框，可以使用常数noGoAway，如此实例所示。

请注意，因为该实例没有使用到参考常数，所以假定0（0×0）是一个填充数。但是，您可能希望使用此常数。例如，您的应用程序可存储代表某个对话框类型的编号，或者存储一个记录的间接指针，该记录可以保留关于对话框或其它窗口类型的状态信息，本书的“窗口管理器”一章对此作了介绍。您可以随时使用窗口管理器的过程SetWRefCon来更改一个对话框的对话记录中的值，并且您可以使用GetWRefCon函数来确定其当前值。

在程序清单6－4中使用的一个应用程序定义的常数可以指定项目列表的资源标识符。在下一节的“为警告栏和对话框提供项目”中介绍了如何创建一个项目列表。

当您希望不定模式对话框或可移动定模式对话框有一个标题时，您可以在对话资源中提供一个文字字符串。您可以为标题栏指定一个不含有文字的空字符串。在实例中指定字符串“Spellcheck Options”是为了使代码段具有可读性，但因为该实例创建的是一个定模式对话框，所以不显示标题栏。

您可以使用对话管理器根据三个标准位置来自动地定位对话框。例如，在程序清单6－4中，指定了alertPositionParentWindow常数将对话框定位在用户正在使用的文稿窗口上。关于这些标准位置的详细说明，请参阅6－62页的“定位警告栏和对话框”。

为警告栏和对话框提供项目

您可以使用一个项目列表（‘DITL’）资源来存储关于警告栏或对话框中所有项目（文字、控制部件、图标、或图片）的信息。如6－18页的“创建示警声和警告栏”和6－23页的“创建对话框”中所介绍的，您可以在警告（‘ALRT’）资源或对话（‘DLOG’）资源中指定项目列表资源的资源标识符。

在一个警告栏或对话框的项目列表资源中，您可以指定其静态文字、按钮和图标的资源标识符及QuickDraw图片。此外，您还可以在一个对话框的项目列表资源中指定复选框、选项按钮、可编辑文字和其它类型的控制部件（如弹出式菜单）的资源标识符。

在图6－13所示的实例中，当用户从苹果菜单中选择About…命令时，应用程序SurfWriter显示出一个警告栏。若要在警告栏的左上角中显示它自己的图标，应用程序可以使用Alert函数。一个资源标识符为128的警告资源将被传送到Alert函数。随后，警告资源指定一个资源标识符为128的项目列表资源。项目列表资源便指定一个“确定”按钮、某些静态文字和资源标识符是128的一个图标。（通常可为项目列表资源及其警告资源或对话资源指派相同的资源标识符，但不是必须这样做。）

在此实例中，当用户选择About命令时，应用程序SurfWriter通过调用Alert函数来显示警告信号。

```
itemHit :=Alert(kAboutBoxID,@MyEventFilter);
```

在此实例中，Alert函数可以显示由kAboutBoxID资源标识符代表的警告资源所定义的警告栏。如6－81页的“响应警告栏中的事件”中介绍的，当显示警告栏时，Alert函数可以处理大多数用户动作。当用户在警告栏中单击某按钮时，Alert便取消警告栏，并且向您的应用程序返回选定按钮的项目编号。在此实例中，所指向的应用程序定义的函数MyEventFiler允许后台应用程序接收其窗口的更新事件。

程序清单6－5表示Rez输入格式的资源，Alert函数用它来显示图6－13所示的警告栏。

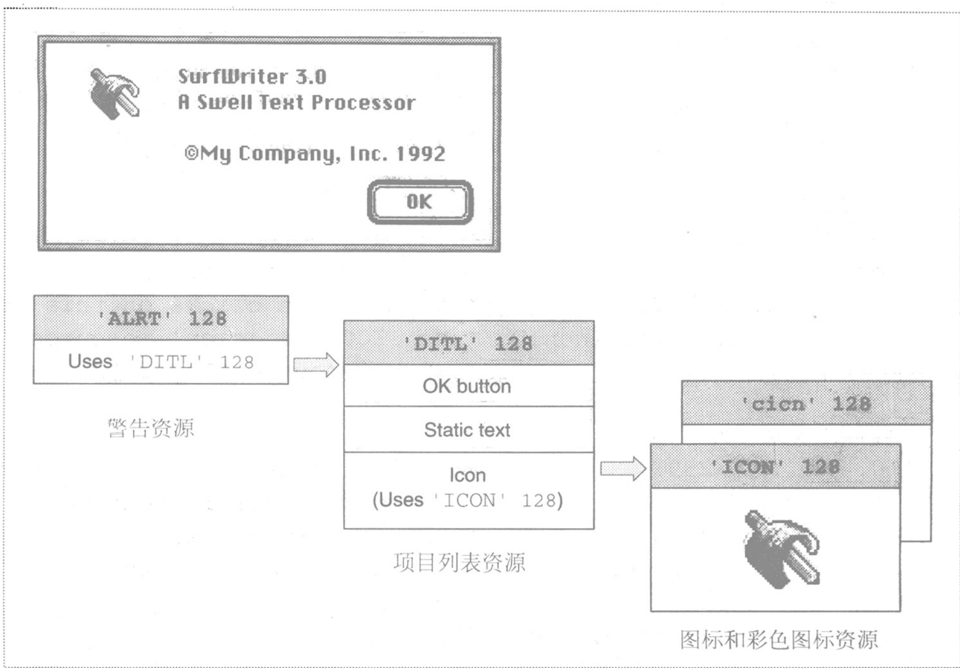


图 6 – 13　各种资源与警告栏的关系

程序清单 6 – 5 用于为警告栏提供项目的 Rez 输入

```
resource 'DLOG' (kSpellCheckID, purgeable) { /*dialog resource*/
    {62, 184, 216, 448},         /*rectangle for dialog box*/
    dBoxProc,                    /*window definition ID for modal dialog box*/
    visible,                     /*display this dialog box immediately*/
    noGoAway,                    /*don't draw a close box*/
    0x0,                         /*initial refCon value of 0*/
    kSpellCheckDITL,             /*use item list with res ID 400*/
    "Spellcheck Options",        /*title if this were a modeless dialog box*/
    alertPositionParentWindow    /*place over document window*/
```

```
resource 'DITL' (kAboutBoxDITL, purgeable)
                                {/*items for About SW alert box*/
         /*ITEM NO. 1*/
     {   {86, 201, 106, 259},      /*display rectangle for item*/
         Button {                  /*the item is a button*/
                  enabled,         /*enable item to return click*/
                  "OK"             /*title for button is "OK"*/
         },
         /*ITEM NO. 2*/
         {10, 20, 42, 52},         /*display rectangle for item*/
         Icon {                    /*the item is an icon*/
                  disabled,        /*don't return clicks on this item*/
                  kSurfWriterIconID /*use 'ICON' & 'cicn' resources */
                                   /* with resource IDs of 128*/
         },
         /*ITEM NO. 3*/
         {10, 78, 74, 259},        /*display rectangle for the item*/
         StaticText {              /*the item is static text*/
                  disabled,        /*don't return clicks on this item*/
                  "SurfWriter 3.0\n"/*text string to display*/
                  "A Swell Text Processor \n\n "
                  "©My Company, Inc. 1992"
         },
         /*ITEM NO. 4*/
         {0, 0, 0, 0},        /*help items get an empty rectangle*/
         HelpItem {           /*invisible item for reading in help balloons*/
              disabled,/*don't return clicks on this item*/
              HMScanhdlg/*scan resource type 'hdlg' for help balloons*/
              {kAboutBoxHelp}    /*get 'hdlg' with resource ID 128*/
         }
     }
};
data 'ICON' (kSurfWriterIconID, purgeable)    {
/*icon data for black-and-white icon for About SurfWriter goes here*/};
data 'cicn' (kSurfWriterColorIconID, purgeable)    {
         /*icon data for color icon for About SurfWriter goes here*/};
```

通常，根据项目在项目列表资源中的位置可以引用项目。例如，当用户在程序清单6－5创建的警告栏中单击“确定”按钮时，Alert函数便返回1。因为“确定”按钮是该列表中的第一个项目，所以对话管理器返回1。(6－77页的“处理警告栏和对话框中的事件”介绍了如何响应由Alert和其它对话管理器函数返回的项目编号。同样，当您使用对话管理器的例程来操纵一个项目时，您可以用项目编号来指定项目，项目编号是对应于该项目在项目列表资源中的位置的一个整数。

重要告诫

项目列表资源通常应标记为可清除的。

在必要时，对话管理器也调用资源管理器来读任一单个项目。

当您创建一个对话框或警告栏时，对话管理器创建一个dialog record（对话记录）。然后，对话管理器读项目清单资源，并且在对话记录的项目域中存储该资源的一个间接指针。因为对话管理器通常拷贝一份项目列表资源并使用该拷贝，所以若干独立的对话框可以共享同一个项目列表资源。根据第6 - 51页开始的“在一个现有的对话框中添加项目”的介绍，您可以使用AppendDITL和ShortenDITL过程来修改或自定一个共享项目列表的若个拷贝，以便在单个对话框中使用。

如果不使用对话资源，您可以直接读对话的项目列表资源，然后向NewDialog（用于创建对话框）传送该资源的一个间接指针及其它信息。但是请记住，您使用对话资源和GetNewDialog函数来本地化您的应用程序更为方便。

以下是一个项目列表资源所包含的各项目信息:

- 一个显示矩形
- 项目类型（在下一节中介绍）
- 一个常数（enadbled或disabled），用来通知对话管理器是否报告该项目发生的事件

  一个文字字符串或一个资源标识符，视项目的类型而定

显示矩形可以确定项目的大小和位置。在警告栏和对话框的局部坐标中指定显示矩形。例如，在程序清单6 - 5中，第一个项目是在坐标（86，201，106，259）指定的一个矩形中显示的，这使得该项目位于此警告栏的右下角。关于显示矩形及其位置的说明，请参阅第6 - 32页的“显示矩形”。

在一个项目列表资源中，您可以指定控制部件、文字、图标、图片和其它由您定义的项目。在程序清单6 - 5中，第一个项目的类型由Button常数指定。下一节将介绍项目类型及其常数。

对于项目列表资源中的每个项目，您还必须通知在项目被单击时向您的应用程序报告。在程序清单6 - 5中，第一个项目是启用的，因为指定了enabled常数。在第6 - 36页的“启用和停用项目”中介绍了何时以及如何启用项目。

根据项目在列表中的类型，您通常应提供该项目的文字字符串或者资源标识符。在程序清单6 - 5中，字符串OK被指定为第一个项目的按钮标题。在6 - 36页的“项目的资源标识符”中介绍了各种类型的项目适合于什么标题和资源。

在后面几节中，详细介绍了您在一个项目列表资源中可以提供的信息。

项目类型

下面的列表说明了您在警告栏和对话框中可以包含的项目类型,以及可以在Rez输入文件中用来指定项目的常数。

常数	说明
Button	一个按钮控制部件
CheckBox	一个复选框控制部件（仅用于对话框）
Control	在‘CNTL’资源文件中定义的控制部件（仅用于对话框）
EditText	一个可编辑文字项目（仅用于对话框）
HelpItem	一个不可视的项目，它使帮助管理器可以将旁述与在项目列表资源 中定义的其它项目相关联
Icon	一个图标，其黑白版本存储在一个‘ICON’资源中，其彩色版本存储在一个‘cich’ 资源中，这两个资源的资源标识符相同
Picture	存储在一个‘PICT’资源中的QuickDraw图片
RadioButton	一个选项按钮控制部件（仅用于对话框）
StaticText	静态文字；即不能进行编辑的文字
UserItem	一个定义应用程序的项目（仅用于对话框）

在《Inside Macintosh工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)*的“帮助管理器”一章中介绍了如何创建和使用HelpItem类型的项目来为您的警告栏和对话框提供旁述。当您指定一个帮助项目时，如程序清单6－5所示，应将它列为列表中的最后一个项目。

在本书的“Finder界面”一章中介绍了图标(‘ICON’)资源和彩色图标(‘cicn’)资源。在《Inside Macintosh绘图》*(Inside Macintosh: Imaging)*中介绍了 ‘PICT’资源。

本书的“控制管理器”一章介绍了如何创建具有一个‘CNTL’资源的控制部件。利用弹出式菜单可以很方便地实现控制部件。6－42页的“弹出式菜单命令项目”介绍了如何将弹出式菜单包含在您的对话框中。

请注意，在警告栏中应该只包含按钮（用户单击该按钮可以取消警告栏）、静态文字、图标和图片。如果需要展示其它项目，您应该创建一个对话框。

警告栏的项目列表资源中第一个项目应该是“确定”按钮；如果需要“取消”按钮，它应该是第二个项目。对话管理器为前两个项目编号提供了这些常数:

```
CONST ok      =1;
   cancel    =2;
```

根据6－18页的“创建示警声和警告栏”中的介绍，您可以在警告资源中定义“确定”或“取消”按钮是否为各警告阶段的系统设定按钮。对话管理器自动地在您指定的按钮周围绘制黑体轮廓线，如果用户按下Return键或者Enter键，对话管理器便作出响应，或者您的事件过滤函数作出响应，这就好像用户单击了系统设定按钮。(6－86页的“为警告栏和定模式对话框编写事件过滤函数”中介绍了事件过滤函数。)

对话管理器不在对话框的系统设定按钮周围绘制黑体轮廓线。6 – 56页的“使用一个定义应用程序的项目来绘制系统设定按钮的黑体轮廓线”说明了您的应用程序应如何在对话框中对系统设定按钮加轮廓线。一般情况下，您应该使每个对话框有一个系统设定按钮—即当用户按下Return或者Enter键时，该系统设定按钮的动作被启动。6 – 86页的“编写用于警告栏和定模式对话框的事件过滤函数”说明了在类似用户单击系统设定按钮的情况下，如何测试这些键按下事件并作出响应。如果您没有提供您自己的事件过滤函数，对话管理器则按项目列表资源中的第一个项目视为系统设定按钮。也就是说，虽然对话管理器不在对话框中的按钮周围绘制黑体轮廓线，但是当用户按下Return键或者Enter键时，对话管理器仍向您的应用程序返回其项目编号。

请不要将系统设定按钮设定成执行危险的动作，例如会导致用户数据丢失的动作。如果所有可能的动作均不安全，就不要在按钮的周围显示系统设定轮廓线，并且提供一个可以忽略Return键或Enter键的事件过滤函数。这样就可以防止用户由于习惯性地按下Return键或者Enter键而意外地破坏了他们的工作。但是，您最好设计一个用户可以用系统设定按钮启动的安全动作，如Save按钮。图6 – 14所示的警告栏可提供一个引起安全动作的系统设定按钮。

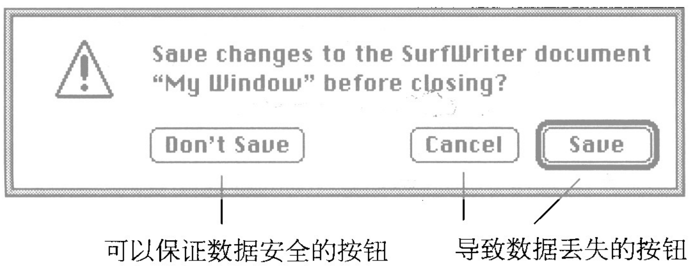


图6 – 14　在警告栏中的一个安全的系统设定按钮

只要有可能，您就应提供一个“取消”按钮，并且在您的事件过滤函数中，将“Command” + “.”键组合及Esc（Escape）键映射到“取消”按钮。

如果您在可编辑文字项目中使用Return键，就不要在任何按钮周围显示黑体轮廓线，因为使用同一个键来做两种不同的事件会使用户感到混乱，而且使得界面缺乏可预见性。

显示矩形

如前面所述，显示矩形可以确定一个警告栏或对话框中某个项目的位置。使用警告栏或对话框的局部坐标可以指定显示矩形。

对于控制部件来说，显示矩形成为控制部件的封闭矩形。若要将控制部件的封闭矩形与它的显示矩形相匹配，那么在控制资源中指定的一个封闭矩形应与您在项目列表资源中指定的显示矩形是恒等的。（在本书的“控制管理器”一章中介绍了控制资源。）

附注

请注意，当一个项目是在一个控制（‘CNTL’）资源中定义了控制部件时，添加到更新区域的矩形就是在控制资源中定义的矩形，而不是在项目列表资源中定义的显示矩形。◆

对于可编辑文字项目来说，显示矩形成为TextEdit目的矩形和它的显示矩形。字绕回可发生在包含多行文字显示矩形内，该矩形中放不下的文字被剪贴掉。对话管理器使用QuickDraw的过程FrameRect在显示矩形的3像素以外绘制一个矩形。关于TextEdit的详细说明，请参阅《Inside Macintosh 文本》*(Inside Macintosh: Text)*的“TextEdit”一章。

对于静态文本项目，对话管理器在显示矩形内文字的方法与绘制可编辑文字项目的方法相同，只是对话管理器不在显示矩形之外绘制框架矩形。

对于大于其显示矩形的图标或QuickDraw图片，对话管理器将图标或者图片缩小到与显示矩形一致。

虽然用于一个定义应用程序的项目的过程可以在项目的显示矩形以外进行绘制，但是这并不是推荐的做法，因为如果对话管理器接收到一个涉及显示矩形以外的区域的更新事件（该区域以内即您绘制定义应用程序的项目的区域），对话管理器将不调用您的绘制过程。

附注

在显示矩形中单击任一处均可被认为是单击该项目。如果显示矩形是重叠的，在重叠区域中的单击则被认为是在单击项目列表资源中出现的第一个项目。◆

在您的应用程序和您所并发的所有应用程序内，您应该根据项目的功能以统一的位置显示项目。在警告栏和大多数对话框中，将“确定”按钮放置在右下角，而将“取消”按钮放置在它的左边。图6 － 15表示警告栏中按钮和文字的推荐位置。

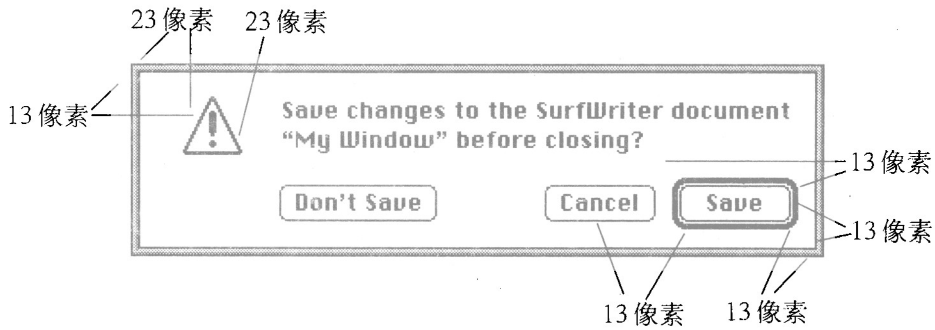


图6 - 15　在警告栏中按钮和文字的统一间距

通常，您应该使用13和23个像素的距离来分隔对话框和警告栏中的项目。(当把系统设定按钮从窗口边缘和其它项目分离开时不必计算黑体轮廓线所占的像素。)但是，请注意，窗口管理器绘制警告栏和定模式对话框时，在其窗口框架内添加3个白像素。因此，当您使用像Rez和ResEdit这样的工具时，请遵循如下说明来指定显示矩形的位置:

- 将右下方按钮的显示矩形放置在距警告栏或右边缘定模式对话框和底边缘10像素的位置上；将其它位于最下方和最右边项目的显示矩形与此按钮对齐
- 将左上方图标（或类似的项目）的显示矩形放置在距警告栏或定模式对话框的顶部边缘和左边缘分别为 10 像素和 20 像素的位置上；将最上方和最左边项目的显示矩形与此项目对齐。当您使用 CautionAlert、NoteAlert 和 StopAlert 函数时，对话管理器自动地将“当心”、“附注”和“停止”警告图标放置在此位置上。当您使用 Alert 函数时，您必须指定图标和位置
- 将警告栏或定模式对话框中的其它元素以13像素或23像素的间距分开，如图6 - 15所示

例如，在图6 - 15中，警告栏矩形的高度被指定为85像素。其 Save 按钮显示矩形的底部坐标是75，其静态文本项目显示矩形的顶部坐标是10。窗口管理器在警告栏的顶部添加3个白像素，在底部添加3个像素，因此，警告栏中 Save 按钮下方和静态文字显示矩形上方分别有13个白像素。程序清单6 - 6说明Rez输入文件如何指定这些显示矩形的位置。

程序清单 6 — 6　使显示矩形的间隔一致的 Rez 输入

```
resource 'DITL'  (200,  purgeable)   {
      {{55,  288,  75,  348},  Button     {enabled,  "Save"},
      {55,  215,  75,  275},  Button     {enabled,  "Cancel"},
      {55,  72,   75,  156},  Button     {enabled,  "Don't Save"},
      {10,  75,   42,  348},  StaticText {disabled,
      "Save changes to the SurfWriter document "^0"before"
                 " closing?"}
      }
};
```

当指定可移动定模式对话框和不定模式对话框中项目的显示矩形位置时，被指定的项目与窗口边缘的全部距离应是 13 或 23 像素。例如，如果图 6 — 15 中的项目是放在不定模式对话框中，那么 Save 按钮显示矩形的顶部坐标应该是 52，而不是 55，其底部坐标应该是 72，而不是 75。

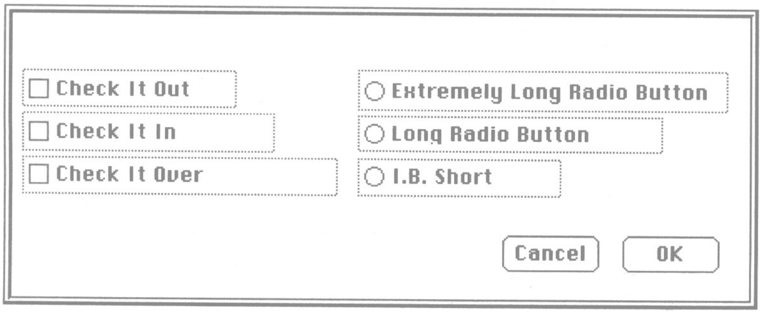
根据前面几节的解释，任何一个按钮都可以是系统设定按钮；使按钮的位置保持一致性是首要的，其次才是指派系统设定按钮。这个规则可以使“确定”按钮和“取消”按钮保持一致的位置。否则，就要根据选择的系统设定按钮来更改按钮的位置。

在警告栏和对话框中，项目的对齐方式依本地化程序的不同而有所区别。虽然在罗马语系系统中，这些项目通常是从左至右对齐，但在阿拉伯或希伯来语系系统中的项目通常是从右至左对齐，因为阿拉伯文和希伯来文是从右至左书写的。在《Inside Macintosh 文本》*(Inside Macintosh: Text)*“TextEdit”一章中介绍的 TextEdit 过程 TESetJust 可以控制界面元素的对齐方式。

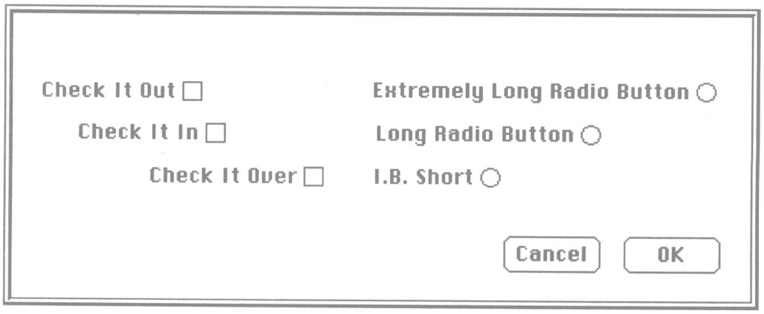
当行的对齐方式是从右至左时　如希伯来文和阿拉伯文)，控制管理器则转置复选框、选项按钮以及它们的标题。也就是说，复选框各选项按钮出现在文字的右边，而不像在罗马语系系统中那样出现在文字左边。因此，当您创建那些需要对齐的复选框、选项按钮和静态文本项目时，要确保它们显示矩形的大小相同。

图 6 — 16 上方所示的对话框有若干复选框和选项按钮，其显示矩形的大小各不相同。在控制管理器将控制部件与它们的标题转置以后，这些项目对齐的结果如图中下一个对话框所示。

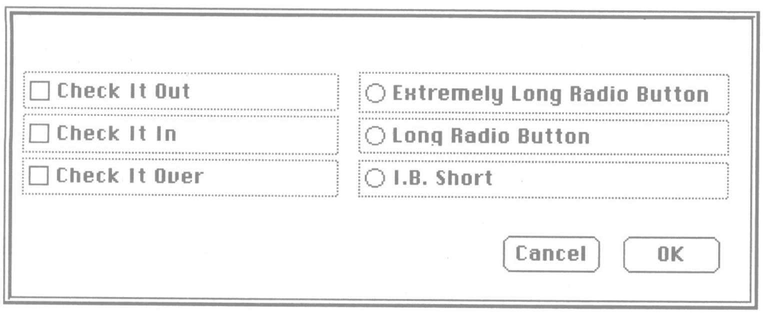
图 6 — 16 下面两个对话框说明当对控制部件及其标题进行转置时，控制管理器如何显示经适当调整大小后的项目。



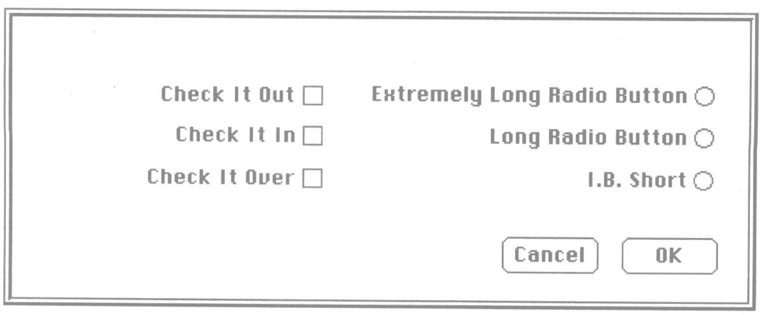
对话框中包含大小不相同的显示矩形



在控制管理器转置复选框和选项按钮与其标题以后的同一对话框



对话框中包含大小相同的显示矩形



在控制管理器转置复选框和选项按钮与其标题以后的同一对话框

图 6 － 16　　经适当调整大小的备用语系系统显示矩形及未经适当调整的情形

启用和停用项目

对于一个项目列表资源中的各个项目，在一个 Rez 输入文件中可包含下列两个常数之一，以便指定当发生涉及该项目的事件时，对话管理器是否应该通知您的应用程序：

常数	说明
enabled	当发生涉及此项目的事件时，通知您的应用程序
disabled	当发生涉及此项目的事件时，不通知您的应用程序

通常，您应该仅启用控制部件，特别是应该启用按钮、选项按钮和复选框，以便它们被用户单击时，您的应用程序能够知道。您通常要停用可编辑文字和静态文本项目。因为只有在用户单击了“确定”按钮后，才可以使用对话管理器的函数 GetDialogText 来读取该项目的信息（程序清单6－12说明了如何使用GetDialog-ItemText 函数读信息。）所以，一般来说您应停用可编辑文字项目。静态文字仅能用来为用户提供信息，而用户不应该单击这些文字。同样，一般来说应停用那些仅用来提供信息的图标和图片；但是，如果您使用一个图标或图片作为像按钮一样可以接收输入的控制部件，您就必须启用该图标或图片。如果您创建一个定义应用程序的项目（如移动指示标记）来显示信息，您一般就要停用该项目。如果您创建一个定义应用程序的项目（如类似按钮的控制部件）来接收输入项，您就必须启用该项目。

请不要将停用一个项目与利用控制管理器的过程 HiliteControl 来使一个控制部件无效相混淆。当您不希望控制管理器响应在控制部件中的单击时，您将是使该控制部件 inactive 使能（有效）；当您不希望对话管理器报告在控制部件中发生的单击时，您将使该控制部件 disabled 不使能（无效）。

控制管理器是通过某种方式（例如，彩色趋于暗淡）来显示一个无效控制的，而对话管理器所显示的启用项目和停用项目却没有视觉上的区别。图 6－17 表示无效控制和有效控制的不同。控制管理器的 Hilitecontrol 过程可用来使无效的 Eject 按钮彩色趋于暗淡。如果用户单击此按钮，控制管理器则不予响应。但是，当用户单击有效的 Desktop 按钮时，控制管理器便使该按钮反色显示 8Tick 时间 。

通常，按钮和其它控制部件是被启用的，停用它们并不改变它们的外观；在图6－17中，启用的选项按钮如果被停用，其外观显示不变。因为在图6－17中的静态文字“Save this document as”不是控制部件，所以对于在该文字中发生的单击应用程序不需要予以响应。因此，应用程序将它停用；但是，如果应用程序将它启用，该静态文字的外观仍是相同的。

项目的资源标识符

项目列表资源中的最后一个元素通常是一个文字字符串或者一个资源标识符。这要依项目的类型而定。



图 6-17　无效控制按钮和停用项目

对图标、QuickDraw 图片、控制部件（除了按钮以外）、复选框和选项按钮可以提供资源标识符。对于图标，提供一个 'ICON' 资源的标识符；对于 QuickDraw 图片，提供一个 'PICT' 资源的标识符；对于一个控制部件（包括弹出式菜单），提供一个 'CNTL' 资源的ID.在 6-27 页的程序清单 6-5 中，资源标识符 128 指定了哪个 'ICON'(和 'CICN' ）资源用于项目列表资源中的第二个项目。

对于按钮、复选框、选项按钮、静态文本项目和可编辑文字项目，可以在项目列表资源中提供一个文字项目的最后一个元素。在下面几节中，介绍了您应该提供的文字准则。

对于您自己的定义应用程序的项目，可以提供一个标题，或者一个资源标识符。在 6 - 57 页的程序清单 6 - 15 中显示了一个包含有一个定义应用程序的项目的项目列表资源。

按钮、复选框和选项按钮的标题

当您在项目列表中指定一个按钮、复选框或选项按钮时，可以提供一个文字字符串（即控制部件的标题）作为项目的最后一个元素。在第 6 - 27 页的程序清单 6 - 5 中，字符串“确定”项目列表资源中的第一个项目指定了按钮标题。

这些项目的标题可以使用大写书名的方式。通常，您将单字标题和多字标题（包含 4 个字母以上的字）大写。像 in、an 或 and 这样的字不大写。在《Apple Publications Style Guide》中规定了标题大写的规则，该书可以从 APDA 得到。

如本书的“控制管理器”一章中所解释的那样，一个复选框的标题应该明确地反应出两个对立的状态，因为复选框应该允许用户将一个特殊设定状态或打开或关闭。两种状态中应表达出同等意义上的对立。如果您不能设计出能够明确表示其对立状态的复选框，那么您最好使用选项按钮。利用选项按钮，您就可以使用两个标题，从而分清两种状态。选项按钮应该能代表相关联的、但不一定是相对立的选项。每个选项按钮的标题中包含一个字或一个短语，用来标志该按钮的作用。请记住，如本书的“控制管理器”一章中所介绍的那样，一组中的选项按钮是相互排它的，每次只可以打开一个按钮。

如果可能的话，命名一个按钮的标题应该使用一个动词，用来说明该按钮所执行的动作。用户一般首先读懂一个警告栏或对话框中的文字，然后根据显示的告诫，如按钮标题或位置进行响应。“存储”、“退出”或“推出磁盘”按钮使得用户可以快速地识别和单击正确的按钮．这些标题比“确定”、“是”和“否”更为清楚明确。如果表示该动作的标题不能精简成一个或两个字，为此可能就要使用“确定”、“取消”或“是”及“否”。如果您使用这些通用的标题，就要保证对话框中的词语能够清楚地说明按钮所启动的动作。图6 － 18是一个带有相应的“确定”和“取消”按钮的对话框。

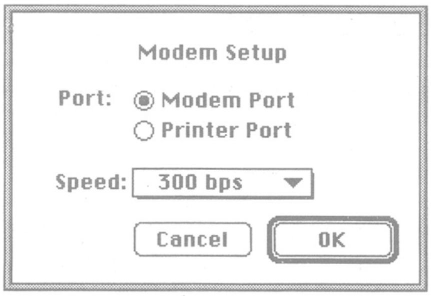


图6 － 18　带有“确定”和“取消”按钮的一个对话框

“取消”的意思是“毫无影响地取消此次操作”而不是说明“我已经读过此对话框”或者“不惜代价的停止您正在做的事情”。当用户单击您的警告栏、定模式对话框和可移动定模式对话框中的“取消”按钮时，您的应用程序就应该撤消自从显示警告栏或者对话框以来发生的所有动作，并且取消该对话框。

当用户单击不定模式对话框中的一个按钮时，您的应用程序不应该取消该对话框而当用户单击现用不定模式对话框的关闭格或者从文件菜单中选择“关闭”时，您就应该取消该对话框。

如果不可能返回到在开始一个操作之前所存在的状态，就不要使用“取消”按钮。您可以使用“停止”或者“确定”这两个可以适用于不同情况的按钮。“停止”按钮可以将一个局部完成的任务的结果原封不动地保留起来，而“取消”按钮总是将应用程序及其文稿还原到它们先前的状态。使用“确定”按钮，可以关闭警告栏、定模式对话框或可移动定模式对话框，并且接受当显示对话框期间所做的任何更改。

因为要撤消从一个不定模式对话框启动的最后一个动作是很困难的，所以这些对话框一般没有“取消”按钮，但它们可能有“停止”按钮。例如，在图6 － 19所示的可移动定模式对话框使用了一个“停止”按钮；单击该按钮可以停止当前的文件拷贝操作，但先前所拷贝的文件将原封不动地保留下来。

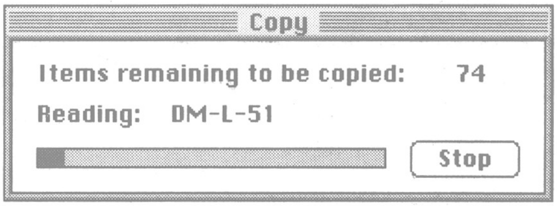


图6 － 19　带有一个“停止”按钮的可移动定模式对话框

在一个需要进行确认的警告栏中，使用的按钮标题应该可以说明当接收了警告栏中的消息以后的结果。例如，如果一个警告栏向您询问是否要“还原到最后一次存储的文稿版 本”，可以使用一个“还原”按钮，而不使用“确定”按钮。请尝试使用在警告消息中使用的同一个动词来作为按钮的标题；如图6 － 20所示的“还原”按钮，就使用了在警告消息中使用的同一个动词。



图6 － 20　带有一个“还原”按钮的一个警告栏

如果警告栏向用户提供的情况是不能有可替换的动作的，则给该框一个标题为“确定”的单一按钮。用户单击此按钮时，您应该解释成“我已经读过此警告栏”。

定模式对话框通常可以结束用户所做的任务。即，当在定模式对话框中进行选择时，用户不能看到所更改的文稿区域。用户只能在撤消该对话框后才可以看到这些更改。如果已更改的部分不合意，用户就不得不重复整个操作。若要向用户提供更好的反馈，您就需要让用户能够看到更改后情况。因此，在定模式对话框中选定任何一项，就应该立即更新文稿的内容，或者，您应该在对话框中提供一个示例区域，用来反映用户的选定项将如何更改文稿。对于立即更改文稿这种情况，“确定”按钮表示“接受此更改”，而“取消”按钮表示“取消在此对话框中所做的所有更改”。

对话管理器使用系统字体来显示按钮的标题（以及所有其它的控制标题）。若要使您的应用程序易于本地化，您就不应该更改此字体。

静态文本项目和可编辑文字项目的文字字符串

对于一个可编辑文字项目，如果您希望该项目仅显示出一个闪烁的光标，就在项目列表资源中指定一个空字符串作为该项目的最后一个元素，或者，如果您希望显示某些系统设定文字信息，就指定一个字符串。

对于一个静态文本项目，当您在项目列表资源中指定它时，可以提供一个文字字符串作为其它的最后一个元素。在程序清单6 - 5的第三个项目中，文字字符 SurfWriter 3.0\nA Swell Text Processor\n\n {SYMBOL 169 \f "Arial Rounded MT Bold"} My company ,Inc.1992 指定了该警告栏的消息。

每当您在警告栏和对话框中提供静态文本项目时，要确保这些消息对用户有意义。请使用简单的、非技术性语言，并且不要提供那些用户不可以响应的面向系统的信息。

如果适用的话，可以在警告栏或对话框中说明文稿或应用程序的名称。例如，在6 - 39页的图6 - 20所示的警告栏中显示了应用程序的名称（SurfWriter）和文稿的名称（MyWindow）。这种消息可以帮助那些同时处理若干文稿或应用程序的用户能够单独对每个文稿或应用程序做决定。6 - 46页的“更改静态文字”中说明了如何使用 ParamText 过程来动态地对您的警告栏和对话框提供文稿窗口的名称。

如果可能的话，可以使用图标和图片。图像比文字更容易说明某些错误情况，人们熟悉的图标更容易帮助用户区分其替换选择。但是，因为经验告诉我们，创建易于理解的或者不防碍所有的国际市场的图标几乎是不可能的，您应该对本地化所使用的任何图标或者图片有所准备。请参阅《Macintosh 用户界面设计原则》*(Macintosh Human Interface Guidelines)* 的“图标”一章中关于创建适当的图标的详细说明。

对于您的静态文本项目，最好要写得文雅一些，即使会使消息显得长一些，也不要使用粗俗的语言。您的消息应该是帮助性的，可能提供一些富有建设性的建议，但不应该像是发号施令。其出发点应该是帮助用户执行任务，而不应是一个有趣的学究式的说明。

当您的应用程序用其它语言来本地化时，这些文字可能变长或者变短。转换后的文字通常比美国英语长50%。您可能需要调整您的显示矩形和您的警告栏及对话框，以便适应转换后的文字。

对话管理器的系统设定设定是使用系统字体来显示静态文字的。若要使您的应用程序易于本地化，您就不应该更改此字体。请不要使用较小的字体，如9点Geneva；像 KanjiTalk等某些语系系统需要12点字体。在您的警告栏和对话框中的所有文字只要使用系统语系就可以方便地进行将来的本地化。

在警告栏中，请尝试把告诉用户如何解决将出现的问题这种信息包括在框中。千万不要让用户考虑其它说明文件中的进一步说明。

停止警告一般用来向用户报告错误。一个好的错误消息可以解释出现了什么错误，为什么发生错误，以及用户如何做等。应该使用用户的语言，而不是用编程语言来表达此信息。

图6－21所示的实例就是一个很差的警告信息－－该信息是用程序员的词汇来表达的，并且没有告诫用户如何处理该问题。



图6－21　一条难于理解且无用的警告信息

图6－22所示的警告消息则稍好一些。虽然没有使用过于专业化的词汇，但却没有提供解决问题的方法。



图6－22　一个稍微好理解一些的警告消息

图6－23所示的警告消息就写得很好。此消息写得具体明确，它使用了非技术性术语来表达，既说明为什么出现错误，又推荐了解决问题的方法。



图6－23　一个清楚且有用的警告消息

若要使一条警告消息易于理解，最好的方法就是仔细分析错误状态本身。不报错，应用程序能否进行处理？该消息是否足够明确，使用户能够纠正错误？建议什么解决方法？

弹出式菜单命令项目

您可以使用弹出式菜单，在一个对话框中向用户提供一个具有相互排它性的选择列表。图6－24图示了在一个对话框中使用弹出式菜单的典型实例。如本书的“控制管理器”一章中所解释的，当用户必须从有许多选项的列表中选择一项或者设定一个特殊值时，或者当您必须提供一个可变选择列表时，弹出式菜单的用途就在于可以用它来替代选项按钮。图6－24所示的弹出式菜单使得应用程序可以让用户根据用户的电脑所使用的调制解调器的不同类型来选择调制解调器的速度。

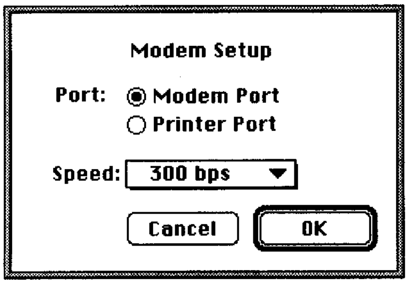


图6－24　在一个对话框中的弹出式菜单

在System 7中，弹出式菜单是作为控制部件来实现的。若要在对话框中显示一个弹出式菜单，您可以

- 在一个使用标准弹出式控制部件定义函数的控制部件中定义弹出式菜单的特殊功能（请参阅本书的“控制管理器”一章中的介绍）
- 像您定义其它菜单中的项目一样来定义一个弹出式菜单的菜单命令项目（使用GetMenu或者NewMenu，请参阅“菜单管理器”一章中的介绍）
- 在对话框的项目列表资源中指定弹出式菜单

对话管理器使用弹出式控制部件定义函数可以自动地绘制弹出框及其阴影，将文字插入到弹出框中，绘制指向下方的三角形和绘制弹出式菜单的标题。当用户将光标移动到一个弹出式菜单上，并且按下鼠标按钮时，弹出式控制部件定义函数便醒目显示该弹出式菜单的标题，显示该弹出式菜单，并且处理所有的用户交互动作，直到用户释放鼠标按钮为止。当用户释放鼠标按钮时，弹出式控制部件定义函数便关闭弹出框，在弹出框中绘制用户所作的选择（或者，如果用户没有新的选择，还原先前的项目），并且取消弹出式菜单标题的醒目显示状态。控制部件定义函数将控制部件的值设定到由用户选定的项目上。您的应用程序便可以使用控制管理器的函数GetControlValue来获取当前选定项目的编号。

若要创建图6 – 24所示的定模式对话框，可以定义一个用来说明该对话框的对话资源和定义用来描述对话项目（包括一个控制部件，其‘CNTL’资源使用标准弹出式控制部件定义函数）的一个项目列表资源。程序清单6 – 7是用于定模式对话框的对话资源和项目列表资源。

程序清单6 – 7 用于定义一个对话框（其中包含有一个弹出式菜单）的对话资源及其项目列表资源的Rez输入

```
resource 'DLOG' (kModemDialog, purgeable) {
     {62, 184, 216, 416}, dBoxProc, visible, noGoAway, 0x0,
     kModemDialogDITL, "", alertPositionMainScreen
};
resource 'DITL' (kModemDialogDITL, purgeable) {
     {    {123, 152, 144, 222}, Button       {enabled, "OK"},
          {123, 69, 144, 139},   Button       {enabled, "Cancel"},
          {13, 70, 33, 204},     StaticText   {enabled, "Modem Setup"},
          {41, 23, 61, 64},      StaticText   {enabled, "Port:"},
          {41, 67, 59, 186},     RadioButton  {enabled, "Modem Port"},
          {59, 67, 77, 186},     RadioButton  {enabled, "Printer Port"},
          {90,18,109,198},       Control  {disabled, kPopUpCNTL},
          {123, 152, 144, 222},  UserItem {disabled}    /*outline OK button*/
          {0,0,0,0},             HelpItem {disabled, HMScanhdlg{kModemHelp}}
                                 /*Balloon Help*/
     }
};
```

程序清单6 – 8中所示的‘CNTL’和‘MENU’用于图 6 – 24 中所示的Speed弹出式菜单。请注意，在项目列表资源中指定的用于控制部件的显示矩形与在控制资源中指定的封闭矩形是同一个。请参阅本书的“控制管理器”中关于如何指定弹出式菜单的控制资源值的详尽说明。

程序清单6 – 8 用于一个弹出式菜单的控制资源和菜单资源的Rez输入

```
resource 'CNTL' (kPopUpCNTL, preload, purgeable) {
     {90, 18, 109, 198},     /*enclosing rectangle of control*/
     popupTitleLeftJust,     /*title position*/
     visible,                /*make control visible*/
     50,                     /*pixel width of title*/
     kPopUpMenu,             /*'MENU' resource ID*/
     popupMenuCDEFProc,      /*pop-up control definition ID*/
     0,                      /*reference value*/
     "Speed:"                /*control title*/
};
```

项目间键盘引导

您的对话框中可能有若干个项目，如可编辑文字项目和滚动列表，它们可以接收来自键盘的输入。您需要给用户提供一个可视的告诫，表示当前哪个项目可以接收来自键盘的输入。每种类型的项目有其自己特殊的指示符。对话管理器自动地在一个可编辑文字项目上显示一个闪烁的光标，表示它正在接收键盘输入。您也可以使用 `SelectDialogItemText` 过程（在6－131页上介绍）来指示在一个可编辑文字项目内选定的文字范围。

当您在滚动列表的周围显示一个有2个黑像素宽的矩形边框（列表与矩形之间有1个白像素宽的距离）时，表示该滚动列表正在接收来自键盘的输入。在图6－25中，在 `Chooser` 对话框中的 `AppleTalkZones` 滚动列表是当前正在接收键盘输入的项目。请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 的“列表管理器”一章中关于在对话框中创建列表的详细说明。

因为所有键入的内容都送到当前活动窗口中，所以在某一时间内只能有一个现用区域和一个指示符。如果在一个对话框中仅有一个元素可以接收键盘输入项，并且该元素就是一个滚动列表，则不需要在该列表的周围放置一个边框。

对话管理器自动地处理鼠标按下事件和 Tab 键的键盘事件。因此，通过单击希望的项目或者通过按 Tab 键来循环于有效项目之中，用户便可以选定可以接收键盘输入的任意项目。当用户按下 `Tab` 键时，对话管理器便接收对当前项目所做的更改，并且选定在项目列表中所列出的下一个项目，以便接收键盘输入项。当用户单击另一个项目时，对话管理器便接收对当前项目所做的更改，并且选定新单击的项目。

操纵项目

在许多情况中，当您在资源文件中定义了警告栏或对话框以后，您不一定对它们做任何更改。但是，如果您希望修改一个项目，您可以使用对话管理器的若干例程来做。

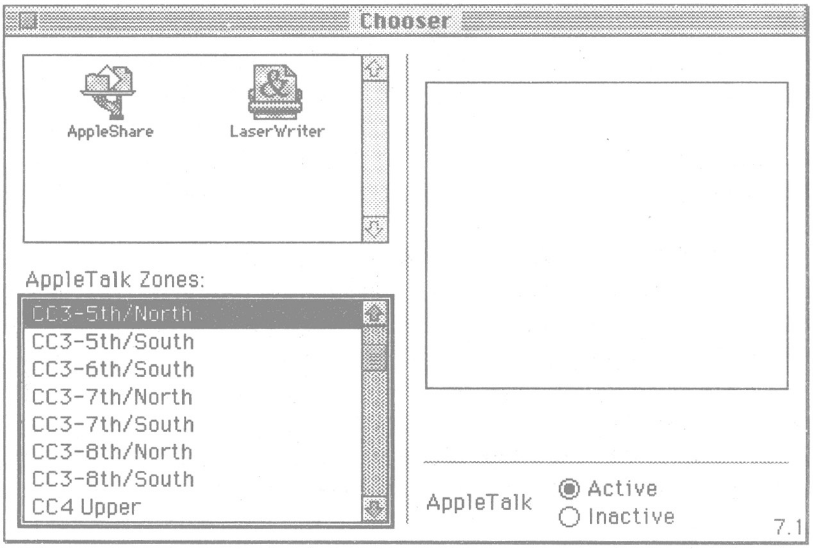


图 6 — 25　　选定的滚动列表

例如，您可以使用ParamText过程来动态地提供警告栏和对话框的文字字符串（如文稿标题）。若要操纵大多数其它类型的项目，您必须首先调用GetDialogItem过程来获取关于该项目的信息。然后，您可以使用其它例程来操纵该项目。例如，您可以使用SetDialogItem过程来更改项目，或者，若要在用户单击“确定”按钮以后，获取用户在可编辑文字项目中键入的文字字符串，您可以使用GetDialogItemText过程。

可用于操纵项目的对话管理器例程小结如下：

例程	说明
AppendDITL	将项目添加到一个对话框中。
CountDITL	计算一个对话框中项目的数量。
FindDialogItem	在一个对话框内查找一个包含一个特定点的项目。
GetAlertstage	返回到发生最后一个警告的阶段。
GetDialogItem	返回在一个对话框中的给定项目的项目类型、显示矩型和控制间接指针或者应用程序定义的过程。
GetDialogItemText	返回给定的可编辑文字或者静态文本项目的文字。
HideDialogItem	隐藏给定的项目。
ParamText	替代静态文本项目中的四个不同的文字字符串。
ResetAlertStage	还原到发生最后一个警告的阶段。

例程	说明
SelectDialogItemText	选定一个可编辑文字项目的文字。
SetDialogItem	设定一个项目的项目类型和显示矩形，或者（对于应用程序定义的项目）一个项目的绘制过程。
ShortenDITL	从一个对话框中取消项目。
ShowDialogItem	重新显示先前由 HideDialogItem 隐藏起来的项目。

下面几节介绍了一些最常用的例程。“更改静态文字”一节中，介绍了使用 ParamText 过程来操纵静态文本项目中的文字。6－48 页的“从可编辑文字项目中获取文字”一节介绍了如何使用 GetDialogItemText 过程来确定在一个可编辑文字项目中用户键入了什么内容。6－51 页的“在一个现有的对话框中添加项目”一节中介绍了如何使用 AppendDITL 过程。6－56 页的“使用定义应用程序的项目来绘制系统设定按钮的黑体轮廓线”一节中介绍了如何使用 SetDialogItem 来定义应用程序的项目。关于前面所列的所有例程的辅助说明，请参阅6－120页的“操纵警告栏和对话框中的项目”和6－129页的“处理警告栏和对话框中的文字”一节。

更改静态文字

如前所述，在一个警告栏或者对话框中说明一个文稿的名称常常是有益的。例如，当用户要关闭一个包含还未存储的更改内容时，应用程序可能就显示出如图6－26所示的一个警告栏。

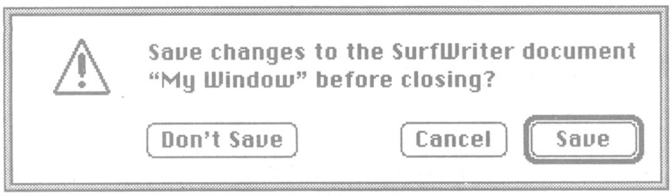


图6－26　显示了一个文稿名称的警告栏

您可以使用 ParamText 过程来动态地向您的警告栏和对话框提供文稿窗口的名称，如程序清单 6-9 中所示的应用程序定义的例程 MyCloseDocument。

程序清单6－9 使用ParamText过程来替代文字字符串

```
PROCEDURE MyCloseDocument (myData: MyDocRecHnd);
VAR
    title:              Str255;
    item:               Integer;
    docWindow:          WindowPtr;
    event:              EventRecord;  {dummy parameter for calling DialogSelect}
    myErr:              OSErr;
BEGIN
    docWindow := FrontWindow;         {point to active window}
    IF (myData^^.windowDirty) THEN
    {document has been changed}
    BEGIN
        GetWTitle(docWindow, title);    {get title of window}
        MyStringCheck(title);
        ParamText(title, '', '', '');   {pass the title in 1st parameter}
        DoActivate(docWindow, FALSE, event);{deactivate the active window}
        item := CautionAlert(kSaveAlertID, @MyEventFilter);
                                        {display alert box}
        IF item = kCancel THEN
            Exit(MyCloseDocument);
        IF item = kSave THEN
            DoSaveCmd;                {save the document}
        myErr := DoCloseFile(myData);
        {close the file}
    END;                              {let click in Don't Save fall through}
    CloseWindow(docWindow);
    DisposPtr(Ptr(docWindow));
END;
```

在此实例中，窗口管理器的函数FrontWindow向当前活动窗口返回一个指针。窗口管理器的另一个函数GetWTitle则返回该窗口的标题。MyCloseDocument例程将此字符串传送到ParamText过程，此过程需要4个文字字符串作为参数。在此实例中，仅需要一个字符串（窗口标题），并且将它传送到第1个参数中；其余3个参数则放置空字符串。

您可以使用ParamText为一个单一警告或对话框提供最多4个文字字符串。在您可以指定文字的任何项目中插入特殊字符^0至^3，您就可以在项目列表中为警告或者对话框指定每个字符串的去向。当您显示警告或者对话框时，ParamText过程动态地用传送到它的第1个参数中的字符串来取代^0，用第2个参数中的字符串来取代^1，以此类推。

重要告诫

若要避免在早于7.1版本的系统软件中出现递归问题，您必须确保不将任何包含有^0至^3字符的字符串传送到ParamText。因此，MyCloseDocument要使用另一个应用程序定义的例程MyStringCheck，将这些字符从传送到ParamText的窗口标题中过滤出来。◆

程序清单6－10是一个项目列表资源的一部分。当应用程序调用CautionAlert时，对话管理器使用先前传送到ParamText过程中的第1个参数来取代文稿窗口的静态文字标题中的字符^0。

程序清单6－10 指定ParamText应该在哪里替代一个警告栏消息中的文字

```
resource 'DITL' (kSaveAlertID, purgeable) {
     {          /*Save button information goes here*/
                /*Cancel button information goes here*/
                /*Don't Save button information goes here*/
          {10, 75, 42, 348},
          StaticText {  /*ask the user to save changes to the document--*/
                    disabled,/* filename inserted with ParamText*/
          "Save changes to the SurfWriter document "^0" before closing?"
          },
                /*help item information goes here*/
     }
```

从可编辑文字项目中获取文字

能够显示出如图6－27所示的不定模式对话框的应用程序在用户点按了“更改”按钮后，便使用GetDialogItem和GetDialogItemText过程。

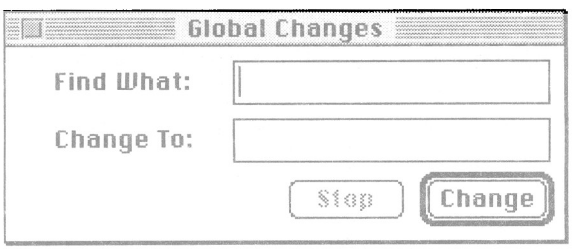


图6－27　在一个不定模式对话框中的两个可编辑文字项目

此对话框告诫用户输入两个文字字符串：一个用来进行搜索，另一个用来取代第1个字符串。程序清单6－11就是用于此对话框的列表资源。在该列表中的第5个项目就是可编辑文字项目，用户在该项目中键入的文字字符串将被用来进行搜索；第6个项目是用户可以键入替代文字字符串的项目。

程序清单6－11 在一个项目列表中指定可编辑文字项目

```
resource 'DITL' (kGlobalChangesDITL, purgeable) {
     {    /*ITEM NO. 1*/
          {70, 213, 90, 271},    Button    {enabled,"Change"},
          /*ITEM NO. 2*/
          {70, 142, 90, 200},    Button    {enabled,"Stop"},
          /*ITEM NO. 3*/
          {10, 23, 27, 98},      StaticText{disabled, "Find What:"},
          /*ITEM NO. 4*/
          {40, 23, 57, 98},      StaticText{disabled,"Change To:"},
          /*ITEM NO. 5*/
          {10, 117, 27, 271},    EditText {disabled, ""},
          /*ITEM NO. 6*/
          {40, 117, 57, 271},    EditText {disabled, ""}
          /*ITEM NO. 7: for drawing outline around Change button*/
          {63, 205, 97, 278},    UserItem {disabled, },
          /*ITEM NO. 8: help item goes here*/
     }
};
```

在程序清单6－12中表示了应用程序程序如何处理单击“更改”按钮事件。（本章后面几节解释了如何处理在不定模式对话框中发生的事件。）

程序清单6－12 在一个可编辑文字项目中获取由用户键入的文字

```
PROCEDURE MyHandleModelessDialogs(theEvent: EventRecord);
VAR
     myDialog:                DialogPtr;
     itemHit, itemType:       Integer;
     searchStringHandle:      Handle;
     replaceStringHandle:     Handle;
     searchString:            Str255;
     replaceString:           Str255;
     itemRect:                Rect;
```

```
BEGIN
{use DialogSelect, then determine whether the event occurred }
{ in the Global Changes dialog box; if so, respond to mouse }
{ clicks as follows}
CASE itemHit OF
    kChange:      {user clicked the Change button}
      BEGIN
        GetDialogItem(myDialog, kFind, itemType,
        searchStringHandle, itemRect);
        GetDialogItemText(searchStringHandle,searchString);
        GetDialogItem(myDialog, kReplace, itemType,
        replaceStringHandle, itemRect);
        GetDialogItemText(replaceStringHandle, replaceString);
        {get a handle to the Stop button}
        GetDialogItem(myDialog, kStop, itemType,
        itemHandle, itemRect);
        {make the Stop button active during the operation}
        HiliteControl(ControlHandle(itemHandle), 0);
        {get a handle to the Change button}
        GetDialogItem(myDialog, kChange, itemType,
        itemHandle, itemRect);
        {make the Change button inactive during the operation}
        HiliteControl(ControlHandle(itemHandle), 255);
        DoReplace(searchString, replaceString);
        {when the operation is complete, dim Stop and make }
        { Change active here}
        END;
    kStop:        {user clicked the Stop button}
        BEGIN
                  {cancel operation, then make Stop button }
                  { inactive and Change button active again}
        END;
  END;
END;
```

在程序清单6－12中，当用户单击“更改”按钮时，GetDialogItem过程返回一个用于包含有搜索字符串的项目的间接指针。因为这是一个用于一个可编辑文字项目的间接指针，所以应用程序可以将该间接指针传送到GetDialogItemText过程，该过程便将该项目的文字字符串返回到它的第二个参数中。这两个过程用于在包含有替换字符串的项目中获取字符串。然后，将这两个字符串传送到一个应用程序定义的例程中，该例程可以用第2个字符串的字符来取代所出现的所有的第1个字符。请注意，当用户单击“更改”时，控制管理器的过程HiliteControl用来使“停止”按钮有效和使“更改”按钮无效——即暗淡的。这就表示，当发生更改操作时，用户可以使用“停止”按钮，但不可以使用“更改”按钮。

在一个现有的对话框中添加项目

使用AppendDITL和ShortenDITL过程，您就可以在一个对话框中动态地添加项目和取消项目。当您创建一个对话框时，对话管理器便创建一个对话记录。对话管理器然后读项目列表资源，并且在对话记录的项目域中存储它的间接指针。因为您创建的每个对话框各有其自己的对话记录，所以您定义的对话框中的项目可以是由同一个项目列表资源定义的。如果若干个对话框共享同一个项目列表资源，并且您希望在适当的时候对各个对话框进行添加或者取消项目，那么，AppendDITL和ShortenDITL过程就尤其有用。

当您调用AppendDITL过程时，您指定一个对话框，并且您指定一个新的项目列表资源，将它增补到对话框现有的项目列表资源中。您还指定对话管理器应该在哪里显示新的项目。您可以使用其中一个常数来设计AppendDITL应该在哪里显示增补的项目：

图6－28是一个现有的对话框和一对要增补的项目。

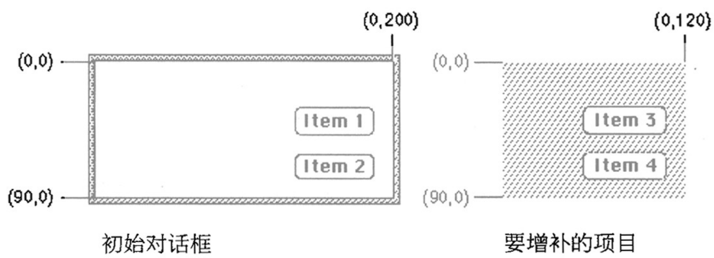


图6－28　一个现有的对话框和要增补的项目

如果您指定了overlayDITL常数，AppendDITL便在对话框上叠加增补的项目。即，AppendDITL将增补项目的显示矩形的坐标（在它们的项目列表资源中指定）解释成对话框中的局部坐标。在图6－28中的对话框上重叠项目后的结果如图6－29所示。

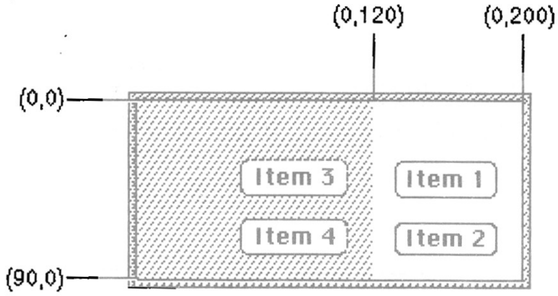


图6－29　重叠项目以后的对话框

如果您指定appendDITLRight常数，AppendDITL便将项目增补到对话框的右边，如图6－30所示，将增补项目的显示矩形放置在相对于对话框的右上坐标的位置上。AppendDITL过程自动地扩展对话框，以便适应新的对话项目。

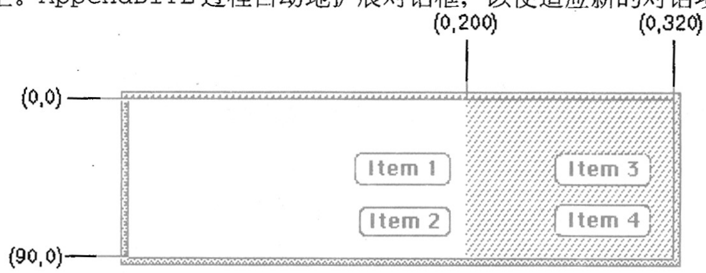


图6－30　将项目增补到右边以后的对话框

如果您指定appendDITLBottom常数，AppendDITL便将项目增补到对话框的底部，如图6－31所示，将增补项目的显示矩形放置在相对于对话框的左下坐标的位置上。AppendDITL过程自动地扩展对话框，以便适应新的对话项目。

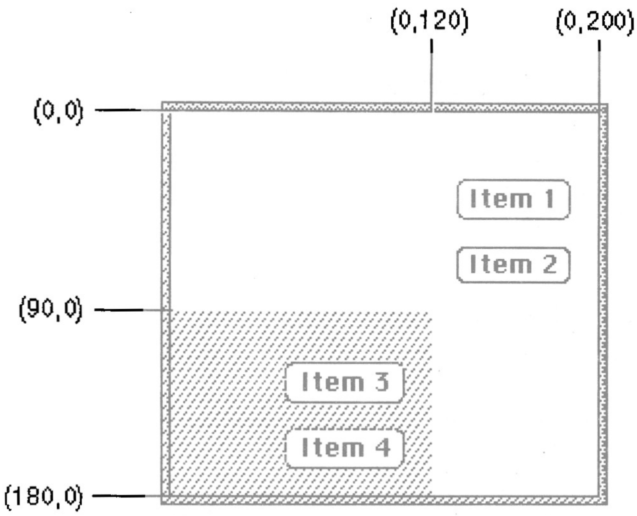


图6 - 31　将项目增补到底部以后的对话框

除了传送overlayDITL，appendDITLRight，或者appendDITLBottom常数这种方法以外，另一种方法是向AppendDITL传送一个负数，这样就可以将项目增补到相对于对话框中现有项目的位置上。此数的绝对值被解释成在对话框中的一个项目，新项目就是相对于此项目来定位。例如，如果您向AppenDITL传送－2，则增补项目的显示矩形就从对话框中第2个项目的左上角进行偏移。在6 - 24页上的图6 - 12所示的一个简单对话框中有两个复选框。如图6 - 32所示，相对于第1个复选框来增补了另一个项目后，在同一个对话框中，在两个现有的复选框之间出现了新的项目。

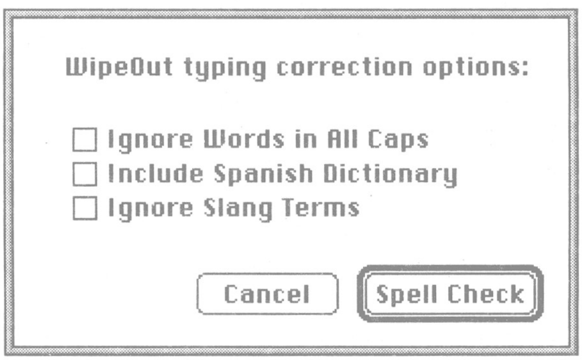


图6 - 32　在对话框中，相对于一个现有的项目增补了一个项目

程序清单6 - 13中所示的应用程序定义的例程叫做DoSpellBoxWithSpanish，该例程使用了AppendDITL过程来添加新的项目。

```
};

FUNCTION DoSpellBoxWithSpanish: OSErr;
VAR
	theDialog:		DialogPtr;
	myNewItem:		Handle;
	docWindow:		WindowPtr;
	event:			EventRecord;
BEGIN
	theDialog := GetNewDialog(kSpellCheckID, NIL, Pointer(-1));
	IF theDialog <> NIL THEN
	BEGIN
		myNewItem := GetResource('DITL', kSpanishDITL);
		IF myNewItem <> NIL THEN
		BEGIN
			AppendDITL(theDialog, myNewItem, kAppendItem);
			{kAppendItem = -3}
			ReleaseResource(myNewItem);
			docWindow := FrontWindow;  {get the front window}
			{if there's a front window, deactivate it}
			IF docWindow <> NIL THEN
				DoActivate(docWindow, FALSE, event);
			ShowWindow(theDialog); {show dialog box with appended item}
			MyAdjustMenus;         {adjust menus as needed}
			REPEAT
				ModalDialog(@MyEventFilter, itemHit);
					{handle clicks in checkboxes here}
			UNTIL ((itemHit = kSpellCheck) OR (itemHit = kCancel));
					{handle clicks in buttons here}
			DisposeDialog(theDialog);
			DoSpellBoxWithSpanish := kSuccess;
		END
		ELSE
			DoSpellBoxWithSpanish := kFailed;
	END
```

DospellBoxWithSpanish例程使用GetNewDialog来创建一个对话框。在程序清单6 - 14中您将会看到，传送到GetNewDialog中的对话资源的资源标识符是402，此对话资源又指定一个资源标识符是402的项目列表资源。然后，DoSpellBoxWithSpanish例程使用资源管理器的函数GetResource来得到一个用于第2个项目列表资源的间接指针；此项目列表资源中包含“Include Spanish Dictionary”复选框。通常在AppendDITL 的最后一个参数中设定值 - 3，DoSpellBoxWithSpanish例程便将第2个项目列表资源中的项目增补到对话框中相对于第3个项目（“Ignore Word in All Caps”复选框）的位置上。程序清单6 - 14是用于该对话框的对话资源、它的常规项目列表资源，以及AppendDITL所添加的项目列表资源。

程序清单 6 － 14 用于一个对话框及其增补项目的Rez输入

```
# define kSpellCheckID 402        /*resource ID for Spell Check dialog box*/
# define kSpellCheckDITL 402      /*resource ID for item list resource*/
# define kSpanishDITL 257         /*resource ID for item list resource to
                                  append*/
# define kAppendHelp 257          /*resource ID for 'hdlg' for appended item*/

resource 'DLOG' (kSpellCheckID, purgeable) { /*Spell Check dialog box*/
    {62, 184, 216, 448},
    dBoxProc,                /*make it modal*/
    invisible,               /*make it initially invisible*/
    noGoAway, 0x0, kSpellCheckDITL, "Spellcheck Options",
    alertPositionParentWindow  /*place over the document window*/
};
resource 'DITL' (kSpellCheckDITL, purgeable) {
    /*items for Spell Check dialog box*/
    /*ITEM NO. 1, the "Spell Check" button, goes here*/
    /*ITEM NO. 2, the "Cancel" button, goes here*/
    /*ITEM NO. 3*/
    {48, 23, 67, 202}, CheckBox {enabled, "Ignore Words in All Caps"},
    /*ITEM NO. 4*/
    {83, 23, 101, 196}, CheckBox {enabled, "Ignore Slang Terms"},
        /*static text, help item, etc. go here*/
};
    /*add this item list resource to Spell Check dialog box only when */
    /* Spanish language dictionary is installed*/resource 'DITL' (kSpanishDITL,
         purgeable) {
    {{18, 0, 36, 209},CheckBox {enabled,"Include Spanish Dictionary"},
    {0,0,0,0}, HelpItem {disabled, HMScanAppendhdlg{kAppendHelp}}/*help*/
    }
};
```

此对话资源指定了对话框是不可视的，因此，在显示对话框以前，应用程序就可以在对话框中添加新的项目。在程序清单6－13中，DoSpellBoxWithSpanish例程使用窗口管理器的过程ShowWindow，在增补了新的项目以后显示该对话框。(6－61页的“显示警告栏和对话框”中详细介绍了如何显示对话框。)

在增补的项目列表资源中包含的一个帮助项目可以使帮助管理器使用除原来与对话框相关的帮助资源以外的，并且与该项目相关的帮助资源。请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh : More Macintosh Toolbox)* 的“帮助管理器”一章中关于使用一个帮助项目中的 HMScanAppendhdlg 标识符的说明。

在程序清单 6－13 中使用了资源管理器的过程 ReleaseResource。AppendDITL 过程可以修改对话框的内容（例如，放大它）。若要在以后可以使用未经修改的对话框版本，您的应用程序就需要使用 ReleaseResource 来释放被增补项目列表所占用的存储器。反之，如果您的应用程序再次调用 AppendDITL 将项目添加到该对话框中，该对话框将保留先前调用的过程所修改的内容。例如，如果您先前使用了 appendDITLBottom 常数，在对话框的底部仍将比较长。

当您可以调用 ShortenDITL 过程从对话项目列表的尾部取消项目时，您指定一个指向对话框的指针和要从项目列表的尾部取消的项目的项目编号。请注意，ShortenDITL 不自动调整对话框的大小，如果您需要调整对话框的大小，您可以使用窗口管理器的过程 SizeWindow。您可以通过调用 CountDITL 函数来确定一个对话框在项目列表资源中有多少个项目。

使用一个定义应用程序的项目来绘制一个系统设定按钮的黑体轮廓线

您可以为对话框定义您自己的项目类型。例如，您可能希望在一个对话框中显示一个时间为当前时间的时钟。您也可以使用定义应用程序的项目在一个对话框中的系统设定按钮周围绘制一个黑体轮廓线。

您不应该在一个警告栏中使用定义应用程序的项目，因为它们使编程增加了不必要的复杂性。如果您需要一个定义应用程序的项目，请使用对话框。

若要定义您自己的项目，就在您的项目列表资源中包括一个类型的项目；该项目应该有一个显示矩形，但没有与之相关的文字及资源标识符。使用此项目列表资源的对话资源必须指定 invisible 常数。这样，当您安装一个用于您的定义应用程序的项目的绘制过程时，用户看不到该对话框。当安装完毕可以绘制定义应用程序的项目的过程后，您可调用窗口管理器的过程 ShowWindow 来显示该对话框。

例如，在图 6－32 所示的对话框中的系统设定按钮（Spell Check）加有轮廓线。若要对按钮加轮廓线，应用程序就必须在该对话框的项目列表资源中添加一个 UserItem 类型的项目。

这样，应用程序定义的绘制过程就可以在“拼写检查”按钮的周围绘制一个边框，在程序清单6－15的项目列表资源中指定了一个用于定义应用程序的项目的显示矩形，它比“拼写检查”按钮的显示矩形更大。

程序清单 6 － 15 用于在一个项目列表中的定义应用程序的项目的 Rez 输入

```
resource 'DITL' (kSpellCheckDITL, purgeable) {
     /*ITEM NO. 1: OK button--the default*/
     {    {123, 170, 144, 254}, Button {enabled,"Spell Check"},
     /*ITEMs 2-5 go here: Cancel button, two checkboxes, and static text*/
     /*ITEM NO. 6: application-defined item*/
          {115, 164, 152, 260},  /*6th item*/
          UserItem {             /*draw procedure for item draws an outline*/
                  disabled,      /*1st item lies inside this--1st is enabled*/
          }
     }
};
```

因为位于定义应用程序的项目之内的“拼写检查”按钮是有效的，所以定义应用程序的项目就是无效的。因为在此项目列表资源中，“拼写检查”按钮被列在定义应用程序的项目之前，所以当用户点按“拼写检查”按钮时，对话管理器便作出报告。但是，请注意，当定义应用程序的项目被启用时，如果用户点按它们，对话管理器则报告它们的项目编号。

附注

虽然用于定义应用程序的项目的绘制过程可以在项目的显示矩形以外进行绘制，但不建议这样做，因为如果对话管理器接收到一个涉及显示矩形以外的区域的更新事件，但您绘制的定义应用程序的项目是在该区域内，对话管理器则不调用您的绘制过程。◆

程序清单6 － 14是用于对话框的对话资源。请注意，在该对话资源中的`invisible`常数指定了该对话框在最初时是不可视的。

您必须提供一个过程来绘制您的定义应用程序的项目。您的绘制过程必须有两个参数：来自对话框的项目列表资源的一个对话指针和一个项目编号。例如，如果您的绘制过程叫做`MyDrawDefaultButtonOutLine`，下面的实例就可以说明该过程：

```
PROCEDURE MYDrawDefaultbuttonOutline (   theDialog£∫DialogPtr;
                                        theItem: Integer) ;
```

参数`theDialog`是一个指向对话框的指针，该对话框中包含有定义应用程序的项目。(如果您的过程可以在多个对话框中进行绘制，此参数则告诉您的过程在哪个对话框中进行绘制。) 参数`theItem`是一个编号，它与一个项目在对话框的项目列表资源中的位置相对应。(如果此过程可以绘制多个项目，此参数则告诉该过程绘制哪一个项目。)

若要安装此绘制过程，可以使用GetDialogItem和SetDialogItem过程。使用GetDialogItem返回一个间接指针，该间接指针用于在项目列表资源中所指定的定义应用程序的项目。然后使用SetDialogItem将一个指向您的绘制过程的指针取代此间接指针。当调用您的绘制过程时，对话管理器将当前端口设定成对话框的图形端口。然后对话管理器便根据需要（例如，当您显示对话框和每当对话管理器接收到对话框的一个更新事件时）调用您的过程来绘制定义应用程序的项目。

程序清单6 - 16表示了如何安装可以绘制黑体轮廓线的过程。在这个列表中，GetDialogItem可以获取一个用于定义应用程序的项目（它是程序清单6 - 15中的项目列表资源的第6个项目）的间接指针。然后，过程指针{SYMBOL 64 \f "Albertus (W1) "} MyDrawDefaultButtonOutLine£®它被强制指向一个间接指针）被传送到SetDialogItem，SetDialogItem将绘制过程设定到对话记录中。

程序清单6 - 16 安装用于一个定义应用程序的项目绘制过程

```
FUNCTION DisplayMyDialog (VAR theDialog: DialogPtr): OSErr;
VAR
    itemType:           Integer;
    itemHandle:         Handle;
    itemRect:           Rect;
    docWindow:          WindowPtr;
    event:              EventRecord;
BEGIN
    {begin by creating an invisible dialog box}
    theDialog := GetNewDialog(kSpellCheckID, NIL, Pointer(-1));
    IF theDialog <> NIL THEN
    BEGIN
        {get a handle to the application-defined item (i.e., userItem)}
        GetDialogItem(theDialog, kUserItem, itemType, itemHandle,
         itemRect);
        {install the drawing procedure for the application-defined item}
        SetDialogItem(theDialog, kUserItem, itemType,
        Handle(@MyDrawDefaultButtonOutline), itemRect);
        docWindow := FrontWindow;       {get the front window}
            {if there's a front window, deactivate it}
        IF docWindow <> NIL THEN
            DoActivate(docWindow, FALSE, event);
        ShowWindow(theDialog);          {display the dialog box}
        MyAdjustMenus;                  {adjust menus as needed}
        DisplayMyDialog := kSuccess;
    END
    ELSE DisplayMyDialog := kFailed;
    {call ModalDialog and handle events in dialog box here}
END;
```

使用窗口管理器的过程 ShowWindow 可以显示先前不可视的对话框。在此实例中，当调用 ShowWindow 时，便在定义应用程序的项目内和 Spell Check 按钮的周围绘制一个黑体轮廓线。

程序清单 6 - 17 所示的过程可以在任意大小和形状的按钮周围绘制黑体轮廓线。此过程可以用来绘制前面实例中的“拼写检查”按钮周围的轮廓线。

程序清单 6 - 17 创建一个可以在系统设定按钮的周围绘制黑体轮廓线的绘制过程

```
PROCEDURE MyDrawDefaultButtonOutline(theDialog: DialogPtr; theItem:
    Integer);
CONST
    kButtonFrameInset =    -4;
    kButtonFrameSize =     3;
    kCntrActivate =        0;
VAR
    itemType:          Integer;       {returned item type}
    itemRect:          Rect;          {returned display rectangle}
    itemHandle:        Handle;        {returned item handle}
    curPen:            PenState;
    buttonOval:        Integer;
    fgSaveColor:       RGBColor;
    bgColor:           RGBColor;
    newfgColor:        RGBColor;
    newGray:           Boolean;
    oldPort:           WindowPtr;
    isColor:           Boolean;
    targetDevice:      GDHandle;
BEGIN
    {get the default button & draw a bold border around it}
    GetDialogItem(theDialog, kDefaultButton, itemType, itemHandle, itemRect);
    GetPort(oldPort);
    SetPort(ControlHandle(itemHandle)^^.contrlOwner);
    GetPenState(curPen);
    PenNormal;
    InsetRect(itemRect, kButtonFrameInset, kButtonFrameInset);
    FrameRoundRect(itemRect, 16, 16);
    buttonOval := (itemRect.bottom - itemRect.top) DIV 2 + 2;
    IF ((CGrafPtr(ControlHandle(itemHandle)^^.contrlOwner)^.portVersion) =
         kIsColorPort) THEN
         isColor := TRUE
    ELSE
         isColor := FALSE;
    IF (ControlHandle(itemHandle)^^.contrlHilite <> kCntrlActivate) THEN
```

```
    BEGIN    {control is dimmed, so draw gray default button outline}
        newGray := FALSE;
        IF isColor THEN
        BEGIN
            GetBackColor(bgColor);
            GetForeColor(fgSaveColor);
            newfgColor := fgSaveColor;
            {get the device on which this dialog box is displayed}
            targetDevice :=
        MyGetDeviceFromRect(ControlHandle(itemHandle)^^.contrlRect);
            {use the gray defined by the display device}
            newGray := GetGray(targetDevice, bgColor, newfgColor);
        END;
        IF newGray THEN
            RGBForeColor(newfgColor)
        ELSE
            PenPat(gray);
        PenSize(kButtonFrameSize, kButtonFrameSize);
        FrameRoundRect(itemRect, buttonOval, buttonOval);
        IF isColor THEN
            RGBForeColor(fgSaveColor);
    END
    ELSE     {control is active, so draw default button outline in black}
    BEGIN
        PenPat(black);
        PenSize(kButtonFrameSize, kButtonFrameSize);
        FrameRoundRect(itemRect, buttonOval, buttonOval);
    END;
    SetPenState(curPen);
    SetPort(oldPort);
END;
```

在程序清单6－17中，使用了GetDialogItem来获取Spell Check按钮，然后当该按钮被激活时，使用若干个QuickDraw例程在按钮的显示矩形周围绘制一个黑色轮廓线。如果该按钮是无效的（即暗淡的），MyDrawDefaultButtonOutLine则绘制一个灰色轮廓线。

在绘制一个灰色轮廓线以前，MyDrawDefaultButtonOutLine要确定该对话框是否使用彩色图形端口。正如6－75页的“在您的警告栏和对话框中加入彩色”中所解释的，通过使用与对话资源相同的资源标识符创建一个对话项目色彩表（‘dctb’），您就可以提供一个带有彩色图形端口的对话框。如果对话框使用了一个彩色图形端口，MyDrawDefaultButtonOutLine便使用彩色QuickDraw的函数GetGray，并根据前台和后台的彩色返回一个混合的灰度。

然后，MyDrawDefaultButtonOutLine 使用此灰度为暗淡的系统设定按钮加轮廓线。反之，MyDrawDefaultButtonOutLine 使用 QuickDraw 的过程 PenPat 在黑白显示器上绘制灰度轮廓线。

显示警告栏和对话框

正如6 - 18页的“创建示警声和警告栏”和6 - 23页的“创建对话框”中所介绍的那样，您通常是以资源的形式定义警告和对话框。若要创建警告或对话框，您可以使用对话管理器的函数——如 Alert 或 GetNewDialog——将来自您的项目列表资源和您的警告资源或对话资源的信息在存储器中结合组成一个数据结构，称为Dialogrecord。每当您的应用程序创建一个警告或对话框时，对话管理器便创建一个对话记录，其数据结构为对话记录类型。

对话管理器在相应的警告阶段自动地显示警告栏；并且还自动地显示那些您在对话资源中指定为可视的对话框。但是您必须使用窗口管理器的例程（如 ShowWindow）显示您在对话资源中指定为不可视的对话框。

当您调用创建一个警告的函数（即 Alert，StopAlert，NoteAlert 或 CautionAlert）时，对话管理器便在相应的警告阶段（在您的警告资源中指定了 Visible 常数）自动地显示警告栏。请不要使用除了 Alert，StopAlert，NoteAlert 和 CantionAlert 函数之外的其它例程来显示警告栏。

如果您在一个对话资源中指定了 Visible 常数，那么，当您调用 GetNewDialog 函数时，对话管理器便立即显示对话框。相反，如果您指定的是 invisible 常数，在您调用 GetNewDialog 时，该对话框最初是不可视的，使用窗口管理器的过程 ShowWindow 便可以显示该对话框。当您需要在显示对话框以前使用 GetDialogItem 和 SetDialogItem 来动态地操纵对话项目时，就可以用这种方法。例如，如果您希望安装一个用于对话框的应用程序定义的绘制过程，您可以在一个对话资源中指定 invisible 常数，将一个参数中的该对话资源的资源标识符传送到GetNewDialog，使用 GetDialogItem 和 SetDialogItem 来安装应用程序定义的绘制过程，安装完毕后调用 ShowWindow 来显示该对话框，如前面6 - 58页上的程序清单6 - 16所示。

您应该总是指定 Pointer (-1) 作为 GetNewDialog 的一个参数，使得显示的对话框成为活动（最前面的）窗口。

您应该执行下列与显示警告栏或对话框有关的任务：

- 指定一个相应的屏幕位置，以便显示警告栏或对话框
- 在显示警告栏或定模式对话框以前释放最前面的窗口（如果有的话）
- 确定您是否已经创建了一个不定模式对话框，如果是，则不用创建新的，而是选择它
- 相应地调整带有可编辑文字项目的定模式对话框的菜单和您希望显示的任何可移动定模式和不定模式对话框

DialogSelect函数使用QuickDraw的过程 SetPort 设定警告栏或对话框为当前图形端口。ModalDialog过程和创建警告栏的函数使用DialogSelect来响应更新和激活事件。您还可以使用DialogSelect响应您的不定模式和移动定模式对话框中的更新和激活事件。为了响应更新事件，您还可以使用UpdateDialog函数，亦能够使对话框成为当前图形端口。在这些情况下，当显示、更新、或激活警告栏和对话框时，一般不需要您的应用程序调用SetPort。请参阅《Inside Macintosh绘图》*(Inside Macintosh: Imaging)* 中关于SetPort的详细说明。

在本章的下面几节中详细解释了有关的问题。

定位警告栏和对话框

正如前面6 - 18页的“创建示警声和警告栏”和6 - 23页的“创建对话框”所介绍的，在每个警告资源和对话资源中您需要指定一个矩形。此矩形的尺寸确定了警告栏或对话框的尺寸。您还可以使矩形的坐标成为可以确定警告栏或对话框位置的全局坐标，或者，您可以让对话管理器根据3个标准位置自动地进行定位。若要在System 7中指定这些标准位置，您的应用程序可以在用于警告资源和对话资源的Rez输入文件中使用下列常数:

常数	说明
alertPositionParentWindow	在最前面的窗口上定位警告栏或对话框
alertPositionMainScreen	在主屏幕上定位警告栏或对话框
alertPositionParentWindowScreen	在包含有最前面的窗口的屏幕上定位警告或对框

如果您的应用程序自己定位其警告栏或对话框，就不要使用这些常数，因为您的代码段与对话管理器之间可能会产生冲突。如果您使用了这些常数，就使用它们指定警告栏和对话框的位置。

下面所示的3个图中的各种警告栏可能出现在用户所用的两种显示器中：一种是显示菜单栏的12英寸显示器（主屏幕），一种是显示文稿窗口的全页显示器。这些图表示了对话管理器如何根据警告资源中指定的位置来放置一个警告栏。

图6 - 33所示的一个警告栏是在用户操作一个文稿造成错误时显示出来的；警告资源指定alertPositionParentWindow常数，此常数告诉对话管理器将警告栏置于最前面的窗口上，因此可显示出窗口的标题栏。此位置对于与最前面的窗口直接相关的警告栏或对话框是合适的。您应该总是将警告栏和对话框定位在用户正在操作的屏幕上。

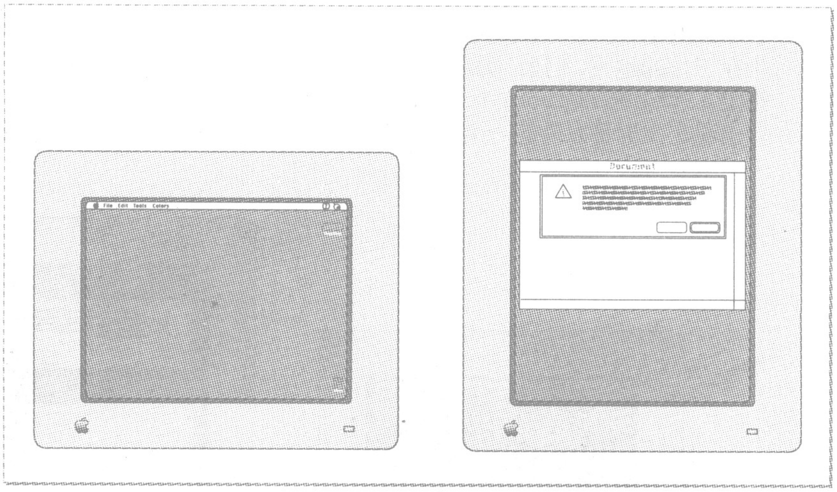


图6 - 33　在一个文稿窗口前面的警告栏

不是所有的警告栏或对话框都与最前面的窗口有联系。有一些可能只与用户在主屏幕上执行的动作有关。例如，图6 - 34给出了当用户选择苹果菜单的“关于”命令时显示的警告栏。对于这样的警告栏或对话框，您应该在警告或对话资源中指定alertPositionMain Screen常数。图6 - 34说明了对话管理器是如何在靠近主屏幕的顶部中心定位这样的警告栏的。

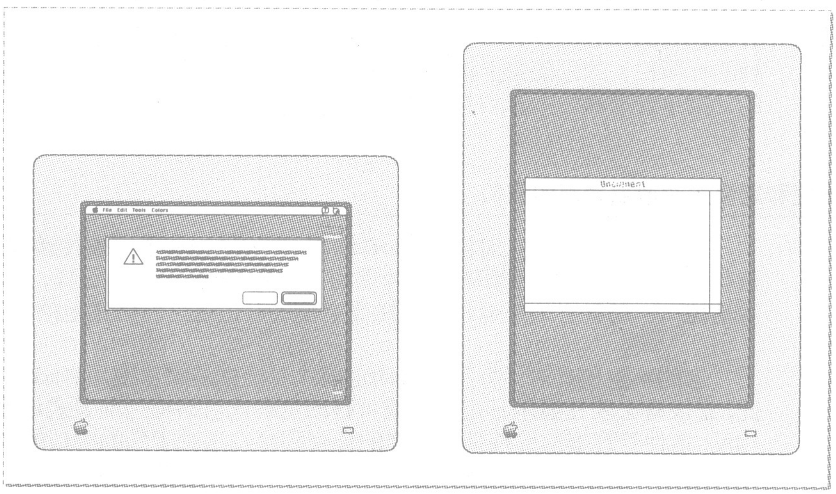


图6 - 34 主屏幕上的警告栏

有时，您可能需要显示一个既不用于最前面窗口也不用于主屏幕上执行的动作的警告栏或对话框。为了引起用户的注意，您应该把这样的警告栏或对话框置于屏幕上用户正在工作的地方。例如，如果由于可用磁盘空间不足而需要向用户警告，那么您就应该指定 alterPositionParentWindowScreen 常数。图 6 - 35 显示了当一个文稿窗口出现在主屏幕以外的一个屏幕上时对话管理器是如何显示这样的警告栏或对话框的。

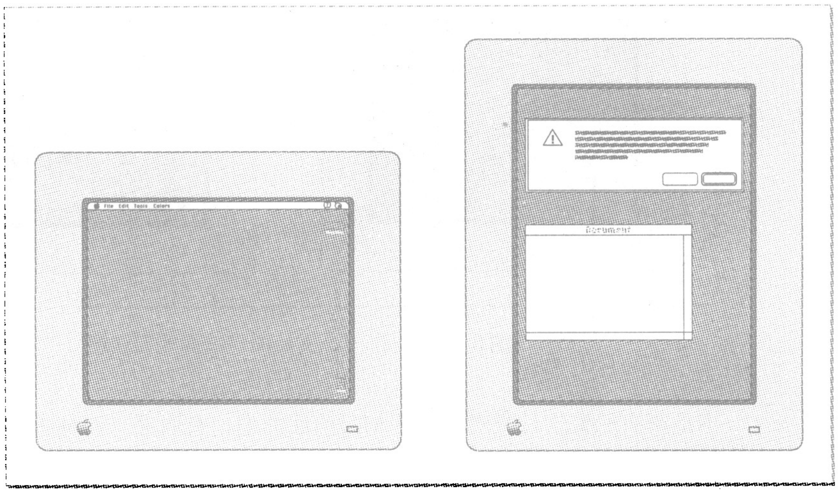


图 6 - 35　位于文稿窗口屏幕的警告位置的警告栏

如果您不指定一个定位常数，对话管理器就使用您的警告资源或对话资源中的矩形坐标作为全局坐标，指定在什么地方放置您的警告或对话框。如果您希望自己用这种方式指定位置，您就应该尽量把警告栏和对话框定位在用户正在工作的屏幕或窗口的左右边缘之间的中心位置。您还应该把警告栏和对话框（包括不定模式和可移动定模式对话框的标题栏）的顶部置于菜单栏的下方。您可以使用 GetMBarHeight 函数（在本书的“菜单管理器”一章中有介绍）来确定菜单栏的高度。

停用警告栏和定模式对话框后面的窗口

对于警告和定模式对话框，ModalDialog 过程在事件被传递到您的事件循环之前捕获所有的事件。事件循环通常为您的窗口处理激活事件。这样，如果一个窗口是当前活动窗口，您就必须在显示警告栏或定模式对话框之前显式地停用该窗口。

您的不定模式对话框和可移动定模式对话框从来不使用 ModalDialog 过程。因此，您没有必要在显示不定模式或可移动定模式对话框之前显式地停用最前面的窗口。相反，事件管理器继续视情况向您的应用程序发送关于您的窗口的激活事件，一般您应在普通事件循环中处理这些事件。（本书的“事件管理器”和“窗口管理器”这两章解释了怎样激活和停用窗口。）

本书开始部分的版面 2 给出了当用户选择苹果菜单中的“关于”命令时一个应用程序显示的警告栏。程序清单 6 - 18 是应用程序定义的例程 ShowMyAboutBox，它是用来显示这个警告栏的。

程序清单 6 - 18 在显示警告栏之前停用前面的窗口

```
PROCEDURE ShowMyAboutBox;
VAR
    itemHit:                Integer;
    docWindow:              WindowPtr;
    event:                  EventRecord;
BEGIN
    docWindow := FrontWindow;
        {get the front window}
    {if there's a front window, deactivate it}
    IF docWindow <> NIL THEN
        DoActivate(docWindow, FALSE, event);
    {then show the alert box}
    itemHit := Alert(kAboutBoxID, @MyEventFilter);
END;
```

ShowMyAboutBox 例程使用窗口管理器函数 FrontWindow。如果 FrontWindow 返回一个有效指针，那么 ShowMyAboutBox 就会在调用 Alert 函数显示警告栏之前调用它的 DoActivate 过程来停用那个窗口。当用户单击“确定”按钮时，警告栏即被取消。然后，事件管理器就向应用程序发送更新事件，以便应用程序能够更新所有相应窗口的内容区域，同时事件管理器还向应用程序发送一个激活事件，以便应用程序能够重新激活以前的最前面窗口。应用程序将在其普通事件循环中处理这些事件。

如果您的应用程序在某警告阶段内没有显示警告栏，那么在停用当前活动窗口之前需使用 GetAlertStage 函数来测试这些阶段。GetAlertStage 函数通过一个 0 至 3 的数返回一个警告的最后发生事件。图 6 - 36 给出了一个警告栏，该警告栏只有在用户连续 3 次重复一个错误时才会出现。

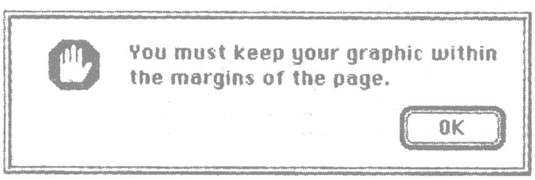


图 6 - 36 只有在第三警告阶段之后才会显示的警告栏

程序清单6 - 19介绍了怎样使用GetAlterStage来确定是否需要在停用文稿窗口之前显示这样的警告栏。

程序清单6 - 19 使用GetAlertStage来确定何时停用当前活动窗口

```
PROCEDURE MyAlert;
VAR
    itemHit:      Integer;
    alertStage:   Integer;
    docWindow:    WindowPtr;
    event:        EventRecord;
BEGIN
    docWindow := FrontWindow;
    alertStage := GetAlertStage;
    IF (alertStage >= 2) AND (docWindow <> NIL) THEN
       {at 3rd alert stage, }
       DoActivate(docWindow, FALSE, event);
      { deactivate front window & }
    itemHit := StopAlert(kStopAlertID, @MyEventFilter);
       { display alert box}
END;
```

显示不定模式对话框

在您创建和显示一个不定模式对话框之前，一定要通过检查来确保该对话框还没有被打开。例如，当用户选择“更改全部”命令时，图6 - 37所示的不定模式对话框就应该出现。在调用这个命令之后，用户可以选定另一个窗口，不定模式对话框即被停用。

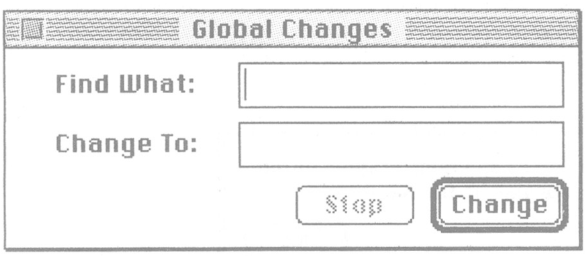


图6 - 37　　一个不定模式对话框，用于更改文稿中的文字

为了避免创建这种对话框的多个版本，每当用户选择“更改全部”命令时，应用程序定义的例程DoGlobalChangesDialog（如程序清单6 - 20所示）应检验对话框是否已经存在。

程序清单 6 － 20 确保创建不定模式对话框之前该对话框还没有打开

```
FUNCTION DoGlobalChangesDialog: OSErr;
BEGIN
    DoGlobalChangesDialog := kSuccess;  {assume success}
    IF gChangeDialogPtr = NIL THEN      {it doesn't exist, so create it}
    BEGIN
    gChangeDialogPtr := GetNewDialog(kGlobalChangesDlog, NIL, Pointer(-1));
    IF gChangeDialogPtr = NIL THEN {handle failure}
        BEGIN
            DoGlobalChangesDialog := kFailed;
            EXIT(DoShowModelessFindDialogBox);
        END;
        {set window refCon to store value that identifies the dbox}
        SetWRefCon(gChangeDialogPtr, LongInt(kGlobalChangesDlog));
    END
    ELSE                {it does exist, so display and select it}
    BEGIN
        ShowWindow(gChangeDialogPtr);   {it's hidden; so show it}
        SelectWindow(gChangeDialogPtr); {bring it to the front}
    END;
    MyAdjustMenus;                      {adjust the menus}
END;
```

在这个实例中，不定模式对话框的指针储存在一个全局变量中。如果全局变量不包含指针，DoGlobalChangesDialog 就使用 GetNewDialog 来创建和绘制对话框。以后，如果用户决定关闭不定模式对话框，应用程序就只把它隐藏起来。这样当用户再需要时，DoGlobalChangesDialog 能在同样的位置显示对话框，并且其选定的文字也与用户最后一次使用它时所选定的一样。在 6 － 94 页的程序清单 6 － 30 中将介绍有关隐藏这个对话框的信息。

如果已经创建了对话框，DoGlobalChangesDialog 就会使用窗口管理器过程 ShowWindow 令对话框可视，并使用 SelectWindow 令其成为当前活动窗口。

最后，DoGlobalChangesDialog 使用应用程序定义的例程 MyAdjustMenus 来调整菜单，使之与不定模式对话框相适应。

6 － 98 页的程序清单 6 － 34 说明应用程序定义的例 DoActivateGlobalChanges-Dialog，它处理这个不定模式对话框的激活事件。DoActivateGlobalChangesDi-alog 例程又使用 DialogSelect。每当用户启用不定模式对话框时，DialogSelect 都会把图形端口设定为不定模式对话框。

针对定模式对话框调整菜单

对话管理器和菜单管理器相互作用，提供对您的菜单栏中的菜单不同程度的存取。对于没有可编辑文字项目的警告栏和定模式对话框，您可以让系统软件为您的菜单栏提供合适的系统存取。

当您的应用程序显示一个警告栏或一个定模式对话框（即dBoxProc类型的窗口）时，会发生这些动作：

1. 系统软件停用帮助菜单中的所有菜单命令项目。但“显示旁述”（或“隐藏旁述”）命令除外，因为该命令是系统软件启用的。
2. 系统软件停用应用程序菜单中的所有菜单命令项目。
3. 如果菜单栏中出现键盘菜单，系统软件就会启用这个菜单但停用“关于键盘”命令。

当您的应用程序显示一个警告栏或为定模式对话框调用ModalDialog过程（6－82页的“响应定模式对话框中的事件”一节中有相关的介绍）时，对话管理器将确定下列这些情况是否属实：

{SYMBOL 341 \f "宋体" \s 10 \h} 您的应用程序没有苹果菜单。

{SYMBOL 341 \f "宋体" \s 10 \h} 您的应用程序有苹果菜单，但在显示对话框时菜单是无效的。

{SYMBOL 341 \f "宋体" \s 10 \h} 您的应用程序有苹果菜单，但在显示对话框时该菜单的第一个命令项目是无效的。

如果这些情况都不属实，系统软件就会采取下列行动：

1. 菜单管理器停用您的应用程序的所有菜单。
2. 如果定模式对话框包含一个可视且现用的可编辑文字域（而且菜单栏包含的一个菜单中的命令有标准的等效组和键：Command-X、Command-C、和Command-V），菜单管理器就启用这3个命令和包含它们的菜单。然后用户就可以使用菜单命令或者它们的等效命令组合键来剪切、拷贝和粘帖文字。（等效命令组合键为Command-X的菜单命令项目必须是前5个菜单命令项目之一。）

当您的应用程序显示没有可编辑文字项目的警告栏和定模式对话框时，它可以放心地允许系统软件处理对菜单栏的存取，如步骤1和2所述。

但是，由于系统软件不能为您处理“取消”或“清除”命令（或任何其它上下文相关命令），为在可编辑文字项目的定模式对话框处理您自己的菜单栏存取，您的应用程序应该执行下列任务：

{SYMBOL 341 \f "宋体" \s 10 \h} 停用苹果菜单或者苹果菜单的第一个命令项目（一般是您的应用程序的 About …命令），以便显示定模式对话框时控制其菜单栏存取 {SYMBOL 341 \f "宋体" \s 10 \h} 停用除编辑菜单外的所有菜单，同时停用编辑菜单中所有不合适的命令 {SYMBOL 341 \f "宋体" \s 10 \h} 使用对话管理器过程DialogCut，Dial-ogCopy，DialogPaste和DialogDelete在可编辑文字项目中支持“剪”、“复制”、“贴”和“清除”命令 {SYMBOL 341 \f "宋体" \s 10 \h}

提供您自己的代码段，用来支持“取消”命令 {SYMBOL 341 \f "宋体" \s 10 \h} 在帮助菜单中启用相应的应用程序项目（系统软件停用“隐藏旁述”/“显示旁述”以外的所有项目）

您不必为系统处理菜单（即应用程序、键盘和帮助菜单）做任何其它事情。系统软件将会为您自动处理这些菜单。

6－132 页介绍 DialogCut、DialogCopy、DialogPaste 和 DialogDelete 过程。在处理编辑菜单中的鼠标按下事件时，您的应用程序可以测试对话框是否是当前活动窗口，然后相应地调用这些例程。

图6－38说明了当显示一个包含可编辑文字项目的定模式对话框时，应用程序是如何停用它自己的所有菜单（它的编辑菜单除外）的。存取编辑菜单将会使那些不愿敲键输入而宁愿拷贝粘贴可编辑文字项目的用户受益匪浅。



图6－38　显示一个定模式对话框时的菜单存取

下面的程序清单6－21给出了一个应用程序定义的例程 MyadjustMenus，Surfwriter 应用程序将在该例程显示一个窗口或对话框之后，调用 ModalDialog 处理定模式对话框中的事件之前，调用 MyadjustMenus 来调整其菜单。当 MyAdjustMenus 确定最前面的窗口是一个含可编辑文字项目的定模式对话框时，它将调用另一个应用程序定义的例程 MyAdjustMenusForDialogs，这个例程要对菜单作适当的调整。下一页的程序清单6－22显示的就是 MyAdjustMenusForDialogs 例程。

程序清单 6 － 21 调整各种窗口中的菜单

```
PROCEDURE MyAdjustMenus;
VAR
    window:                 WindowPtr;
    windowType:             Integer;
    menu:                   MenuHandle;
BEGIN
    window := FrontWindow;
    windowType := MyGetWindowType(window);
    CASE windowType OF
    kMyDocWindow:   {document window is in front}
        MyAdjustMenusForDocWindows;
    kMyDialogWindow:{a dialog box is in front}
        MyAdjustMenusForDialogs;
    kDAWindow     :{adjust menus accordingly for a DA window}
        MyAdjustMenusForDA;
    kNil: {there isn't a front window}
        MyAdjustMenusNoWindows;
    END; {of CASE}
    DrawMenuBar;      {redraw menu bar}
END;
```

程序清单 6 － 22 中的 MyadjustMenusForDialogs 例程，首先确定对话框的类型：定模式、可移动定模式、不定模式。对于定模式对话框，MyAdjustMenus-ForDialogs 停用苹果菜单，使得应用程序可以脱离对话管理器控制它的菜单。然后，MyAdjustMenusForDialogs 例程使用菜单管理器例程 GetMenuHandle 和 DisableItem 停用除编辑菜单以外的所有其它应用程序菜单。[为了提供解释为什么用户不能使用这些菜单的旁述，MyAdjustMenusForDiawgs 使用帮助管理器过程 HMSetMenuResID 给这些菜单重新指派帮助资源；请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh : More Macintosh Toolbox)* 一书中的“帮助管理器”一章，了解更多的信息。]

程序清单 6 － 22 针对一个含有可编辑文字项目的定模式对话框，使菜单无效

```
PROCEDURE MyAdjustMenusForDialogs;
    VAR
        window:                 WindowPtr;
        windowType:             Integer;
        myErr:                  OSErr;
        menu:                   MenuHandle;
BEGIN
    window := FrontWindow;
```

```
windowType := MyGetWindowType(window);
    CASE windowType OF
        kMyModalDialogs:
        BEGIN
            menu := GetMenuHandle(mApple); {get handle to Apple menu}
            IF menu = NIL THEN
                EXIT(MyAdjustMenusForDialogs);
            DisableItem(menu, 0);{disable Apple menu to get control of menus}
            myErr := HMSetMenuResID(mFile, kFileHelpID);
            {set up help balloons}
            menu := GetMenuHandle(mFile);  {get handle to File menu}
            IF menu = NIL THEN
                EXIT(MyAdjustMenusForDialogs);
            DisableItem(menu, 0);          {disable File menu}
            myErr := HMSetMenuResID(mFile, kFileHelpID);
            {set up help balloons}
            IF myErr <> NoErr THEN
                EXIT(MyAdjustMenusForDialogs);
            menu := GetMenuHandle(mTools); {get handle to Tools menu}
            IF menu = NIL THEN
                EXIT(MyAdjustMenusForDialogs);
                DisableItem(menu, 0);      {disable Tools menu}
            myErr := HMSetMenuResID(mTools, kToolsHelpID);
            {help balloons}
            IF myErr <> NoErr THEN
                EXIT(MyAdjustMenusForDialogs);
            MyAdjustEditMenuForModalDialogs;
        END;     {of kMyModalDialogs CASE}
        kMyGlobalChangesModelessDialog:
            ;    {adjust menus here as needed}
        kMyMovableModalDialog:
            ;    {adjust menus here as follows: }
                 { disable all menus except Apple, then }
            { call MyAdjustEditMenuForModalDialogs for editable text items}
    END; {of CASE}
END;
```

为了调整编辑菜单的项目，MyAdjustMenusForDialogs 调用另一个应用程序定义的例程 MyAdjustEditMenuForModalDialogs，程序清单 6 — 23 中给出了这个例程。

MyAdjustEditMenuForModalDialogs 例程使用应用程序定义的代码段来取消命令；在适当时候，使用菜单管理器过程 EnableItem 启用“剪”、“复制”、“贴”和“清除”命令；同时停用支持版本管理器功能的命令。记住，您的应用程序应该使用对话管理器过程 DialogCut，DialogCopy，DialogPaste 和 DialogDelete 在可编辑文字项目中支持“剪”、“复制”、“贴”和“清除”命令。

程序清单 6 － 23 为定模式对话框调整编辑菜单

```
PROCEDURE MyAdjustEditMenuForModalDialogs;
VAR
    window:             WindowPtr;
    menu:               MenuHandle;
    selection, undo:    Boolean;
    offset:             LongInt;
    undoText:           Str255;
BEGIN
    window := FrontWindow;
    menu := GetMenuHandle(mEdit);   {get a handle to the Edit menu}
    IF menu = NIL THEN              {add your own error handling}
        EXIT (MyAdjustEditMenuForModalDialogs);
    undo := MyIsLastActionUndoable(undoText);
    IF undo THEN  {if action can be undone}
    BEGIN
        EnableItem(menu, iUndo);
        SetMenuItemText(menu, iUndo, undoText);
    END
    ELSE          {if action can't be undone}
    BEGIN
        SetMenuItemText(menu, iUndo, gCantUndo);
        DisableItem(menu, iUndo);
    END;
    selection := MySelection(window);
    IF selection THEN
    BEGIN         {enable editing items if there's a selection}
        EnableItem(menu, iCut);
        EnableItem(menu, iCopy);
    END
    ELSE
    BEGIN         {disable editing items if there isn't a selection}
        DisableItem(menu, iCut);
        DisableItem(menu, iCopy);
    END;
    IF MyGetScrap(NIL, 'TEXT', offset) > 0 THEN
        EnableItem(menu, iPaste)  {enable if something to paste}
    ELSE
        DisableItem(menu, iPaste);{disable if nothing to paste}
    DisableItem(menu, iSelectAll);
    DisableItem(menu, iCreatePublisher);
    DisableItem(menu, iSubscribeTo);
    DisableItem(menu, iPubSubOptions);
    END;
END;
```

请参阅本书“菜单管理器”一章了解关于菜单和菜单栏的更多信息。

当用户消除警告栏或定模式对话框时，菜单管理器将把所有菜单还原至警告栏或定模式对话框出现以前它们所处的状态（除非您的应用程序处理它自己的菜单栏存取，这种情况下您必须把菜单还原成它们以前的状态）。您可以用一个类似于MyAdjustMenus（6 - 70页的程序清单6 - 21给出了这个例程）的例程，根据成为当前活动窗口的类型相应地调整菜单。

针对可移动定模式和不定模式对话框调整菜单

尽管系统软件总是让帮助、键盘和应用程序菜单以及它们的命令处于有效状态，但当您显示可移动定模式和不定模式对话框时，它并不做任何别的事情来管理菜单栏。而您的应用程序就应该根据上下文相应地允许或拒绝存取您的其它菜单。例如，如果您的应用程序为一个“搜索与替换”命令显示一个不定模式对话框，您就应该允许存取编辑菜单以帮助用户处理可编辑文字项目，您还应该允许使用文件菜单使用户能打开另一个文件进行搜索。但如果在当前不定模式对话框中不能使用其它菜单的命令，您就应该停用这些菜单。

当创建不定模式对话框时，您的应用程序应该执行下列任务:

- 仅仅停用其命令在当前上下文无效的那些菜单
- 如果不定模式对话框含有可编辑文字项目，就使用对话管理器过程DialogCut、DialogCopy、DialogPaste和DialogDelete在可编辑文字项目中支持“剪”、“复制”、“贴”和“清除”命令

当您的应用程序创建一个可移动定模式对话框时，它应该执行下列任务:

- 启用苹果菜单以便用户能够用它打开其它应用程序
- 如果您的可移动定模式对话框包含可编辑文字项目，则应保持编辑菜单呈现用状态，但需 要使用对话管理器过程DialogCut，DialogCopy，DialogPaste和DialogDelete来支持“剪”、“复制”、“贴”和“清除”命令
- 停用您的所有其它菜单

6 - 70页的程序清单6 - 21给出了一个应用程序定义的例程MyAdjustMenus，SurfWriter将在它显示一个窗口或对话框之后使用这个过程调整其菜单。您可以根据给定的当前活动窗口、可移动定模式对话框或不定模式对话框的性质，使用一个类似的过程相应地调整您的菜单。

显示多个警告栏和对话框

一般情况下，您一次只应向用户展示一个定模式对话框或警告栏。有时，您可能需要同时在屏幕上展示一个定模式对话框和一个警告栏。例如，当用户用与另一个文件相同的文件名称存储一个文件时，标准文件软件包会在标准文件对话框的顶部显示一个警告栏。警告栏询问用户是否覆盖现有文件。

要避免在响应用户动作时关闭一个定模式对话框又立即显示另一个定模式对话框或警告栏。这种情况会导致一种“隧道定模式对话框”效果，用户将感到不知所措。以前的定模式对话框中的内容消失且不能复原，用户很难预测下一步将发生什么情况。

但是，决不能让用户在屏幕上同时看见一个以上的定模式对话框和警告栏。不过您可以同时展示多个不定模式对话框，就像您可以展示多个文稿窗口一样。

当您取消一个覆盖以前警告栏的系统设定按钮的警告栏或定模式对话框时，对话管理器并不重新绘制这个按钮的黑体轮廓线。因而如果您需要显示另一个叠加的警告栏或对话框，您就不应使用警告栏。您应该创建一个定模式对话框，并且您必须给它提供一个定义应用程序的项目，围绕系统设定按钮绘制黑体轮廓线。在一个更新事件之后，`ModalDialog` 过程便令该项目重新绘制。

在 System 7 中，当您停用一个对话框而显示一个警告栏、另一个定模式对话框或一个窗口时，窗口管理器会自动淡化该对话框窗口框架的色彩显示。当您停用一个对话框时，您应该利用控制管理器过程 `HiliteControl` 使对话框的控制部件变成无效状态。您还应该用灰色（而不是黑色）绘制停用窗口系统设定按钮的轮廓线。6 - 58 页的例表 6 - 16 给出了一个应用程序定义的过程，当系统设定按钮变成无效状态时它绘制一个灰色轮廓线；6 - 98 页的程序清单 6 - 34 说明了如何使用 `HiliteControl` 响应对话框的激活事件，使按钮变成无效和有效状态。

从后台显示警告栏和对话框

如果您的应用程序在后台运行（即用户看不到）时，您需要显示一个警告栏或定模式对话框，那么您就应该使用通知管理器给用户发送一个通知。如果您的应用程序执行很长的后台任务，如打印许多文稿或向另一台电脑传送大量数据，当这个操作完成时您可能希望通知用户。在这些情形中，您应该发送一个通知请求，告诉用户该操作什么时候完成。然后通知管理器就会自动地显示包含您所指定的消息的警告栏 您自己没有必要使用对话管理器来创建警告栏。

注意，对话管理器能够提供从您的应用程序到用户的单向通讯通道。当您的应用程序在后台时没有任何措施能把信息从用户传送给应用程序（尽管您的应用程序有可能确定通知是否被接收）。如果您需要向用户征求信息，使用通知管理器要求用户把您的应用程序送到前台。这样用户就可以响应您的警告栏或定模式对话框。请参阅《Inside Macintosh 进程》*(Inside Macintosh : Processes)* 的“通知管理器”一章了解关于通知管理器的信息。

在您的警告栏和对话框中加入彩色

在彩色显示器上，对话管理器会自动将色彩添加至您的警告栏和对话框，使得它们与系统使用的窗口、警告栏和对话框的彩色匹配。从黑白显示器到8位彩色显示器，这些色彩在所有显示器上均能提供了美观一致的显示效果。例如，在一个彩色显示器上，不定模式对话框的标题栏中的跑道线是灰色的，关闭格和窗口窗架是彩色的，而按钮的文字则是黑色。

当您创建警告栏和对话资源时，您的应用程序的警告栏和对话框使用系统的系统设定色彩。一般情况下，您创建警告和对话框所需做的一切只是为您的警告栏和对话框提供色彩。但下列情况除外：

- 当您需要在警告或对话框中包括一个图标的彩色版本时，您必须创建一个 'cicn' 类型的资源，其资源标识符与项目列表资源指定的黑白 'ICON' 资源的资源标识符相同。本书前面的版面 2 给出了包含一个彩色图标的警告栏

- 如果您使用GetNewDialog或NewDialog来创建对话框，并且您需要产生一种混合灰色用来绘制无效（即变暗的）系统设定按钮的轮廓线，您就必须创建一个资源标识符与对话资源相同的对话项目色彩表（ 'dctb' ）资源

6 – 65页的“使用定义应用程序的项目绘制系统设定按钮的黑体轮廓线”一节解释了怎样创建一个为对话框的系统设定按钮绘制轮廓线的绘图例程。如果您停用一个对话框，您应该使它的按钮色彩变淡并使用灰色为系统设定按钮绘制轮廓线。由于GetNewDialog和NewDialog为对话框提供黑白图形端口，您可为对话框创建一个对话项目色彩表资源，强制对话管理器提供一个彩色图形端口。然后您可用混合灰色绘制系统设定按钮的轮廓线。（NewColorDialog 提供一个彩色图形端口。）

甚至当您为绘制灰色轮廓线创建对话项目色彩表资源时，您也不应该更改系统的系统设定色彩。程序清单6 – 2给出了一个对话项目色彩表资源，它没有对系统设定色彩作任何变动，但强制对话管理器供给一个彩色图形端口。

程序清单 6 – 24 使用系统设定色彩的对话色彩资源的Rez输入

```
data 'dctb' ( kGlobalChangesDialog, purgeable ) {
      $ "0000 0000 0000 FFFF°±/* use default colors */
       };
```

使用系统设定的色彩能够确保应用程序界面与Finder及其它应用程序界面保持一致。但如果您需要打破这种一致性，您可以通过对话管理器获得指定系统设定的色彩以外的其它色彩。不管怎样，要意识到在您的警告栏和对话框中的非标准色彩最初有可能迷惑您的用户。

还要意识到不管您作了什么更改，用户都可以通过更改色彩控制板的设定状态来任意改变警告栏和对话框的色彩。

您的应用程序可以在警告色彩表（ ‘actb’ ）资源中指定它自己的彩色，这个资源的资源标识符与警告资源的资源标识符相同；或者在对话项目色彩表（‘dctb’ ）资源中指定它自己的彩色，这个资源具有和对话资源相同的资源标识符。以上这两个资源的格式与窗口色彩表（ ‘wctb’ ）资源的格式一模一样，在本书“窗口管理器”一章中有关于窗口色彩表资源的介绍。

警告

由于运行System 7以前的系统软件版本的电脑并不支持彩色对话框、彩色项目和彩色图标，如果您希望保持向下兼容性就不要给这些元素指定彩色。◆

您没有必要调用任何新例程来更改警告和对话框中使用的彩色。例如，当您调用GetNewDialog函数时，对话管理器将自动装入与对话资源具有相同的资源标识符的对话项目色彩表资源。

同样，通过创建一个与项目列表资源具有相同的资源标识符的项目色彩表（‘ictb’）资源，您可更改控制部件的系统系统设定色彩，更改警告栏或对话框中使用的文字的彩色、字形、字体、和大小。您不必调用任何例程来创建彩色项目。当您使用GetNewDialog函数时，对话管理器首先寻找与项目列表资源具有相同资源标识符的项目色彩表资源。

附注

即使项目色彩表资源不包含任何实实在在的彩色信息并且它仅说明静态文字和可编辑文字项目更改情况，如果您要为警告栏或对话框提供项目色彩表资源，您必须创建警告色彩表资源或对话项目色彩表资源。您不能在并不支持Color QuickDraw的电脑中使用项目色彩表资源设定字体。同时，要意识到更改系统设定字体会使您的应用程序更难实现本地化。◆

即使您提供自己的‘dctb’、‘actb’ 或 ‘ictb’ 资源，您也没有必要为了使用这些资源而去测试您的应用程序是否运行在支持Color QuickDraw的电脑上。

处理警告栏和对话框中的事件

下面两节解释了对话管理器如何使用控制管理器自动处理控制部件中的事件，以及如何使用 TextEdit 自动处理可编辑文字项目中的事件。在“响应控制部件中的事件”和“响应可编辑文字项目中的事件”这两部分中的信息，适用于所有的警告栏和所有的对话框类型：定模式对话框、不定模式对话框和可移动定模式对话框。

若要显示和处理警告栏中的事件，您可以使用对话管理器函数 Alert，NoteAlert、CautionAlert 和 StopAlert。对话管理器处理由用户生成的所有事件，直到用户单击按钮（一般是“确定”或“取消”按钮）为止。当用户单击按钮时，警告栏函数反色显示被单击的按钮，关闭警告栏，并向您的应用程序报告用户单击的对话项目号。您的应用程序负责执行与这个按钮相关联的相应动作。6 - 81 页的“响应警告栏中的事件”一节对此进行了详细的说明。

对于定模式对话框，您可以使用 ModalDialog 过程。对话管理器处理大部分用户对话，直到用户选定一个项目为止。然后，ModalDialog 过程报告用户选定了一个有效项目，您的应用程序负责执行与那个项目相关联的动作。您的应用程序总是反复调用 ModalDialog，响应由 ModalDialog 报告的对有效项目的鼠标单击事件，直到用户单击“确定”或“取消”按钮为止。6 - 82 页的“响应定模式对话框中的事件”一节对此作了详细介绍。

对于警告栏和定模式对话框，您应该提供一个事件过滤函数作为警告栏函数或 Modal Dialog 过程的一个参数。当用户与警告栏或定模式对话框对话时，这些例程在处理每个事件之前把事件传递给您的事件过滤函数。您的事件过滤函数可以处理对话管理器不处理的所有事件，如果有必要，它还可以选择处理一般由对话管理器处理的事件。6 - 86 页的“编写警告栏和定模式对话框的事件过滤函数”一节对此作了详细介绍。

若要处理不定模式或可移动对话框中的事件，您可以使用 IsDialogEvent 函数来确定事件是否发生在对话框中。对于对话框中发生的每个事件类型（包括空事件），IsDialogEvent 返回 TRUE；否则返回 FALSE。当 IsDialogEvent 返回 TRUE 时，您可以使用DialogSelect函数自动处理可编辑文字项目中的键按下事件，自动处理更新和激活事件，报告用户用鼠标单击的有效项目。然后相应地响应现用项目中的鼠标单击事件。

另一方面，您可以像处理其它窗口中的事件一样处理不定模式和定模式对话框中的事件。这就是说，当您接收到一个事件时，您可以首先确定所发生的事件的类型，然后根据当前窗口的类型而采取相应的动作。如果不定模式或可移动定模式对话框在前面，您就可以提供执行这个对话框的所有特定动作的代码段，并调用DialogSelect函数处理您的代码段不处理的所有事件。6 - 89 页的“响应不定模式和可移动定模式对话框中的鼠标事件”一节，6 - 94页的“响应不定模式和可移动定模式对话框中的键盘事件”一节，以及6 - 97页的“响应不定模式和可移动定模式对话框中的激活和更新事件”一节，都采用了这种替代办法。

响应控制部件中的事件

对话管理器大大简化了执行按钮、复选框、弹出式菜单和选项按钮需做的工作。对于警告栏和所有类型的对话框（定模式、不定模式和可移动定模式），对话管理器可以使用控制管理器例程自动显示控制部件，相应地醒目显示控制部件，当鼠标按下事件发生在控制部件中时向您的应用程序报告。例如，当用户把光标移到一个有效按钮上并按下鼠标按钮时，对话管理器使用控制管理器函数 TrackControl 反色显示这个按钮。当有效按钮仍处于反色显示状态用户释放鼠标按钮，对话管理器使用 TrackControl 来报告哪个项目被单击。您的应用程序就相应地作出响应。例如，执行与“确定”按钮相关联的操作，当一个选项按钮被单击时对其它所有选项按钮解除选定，或者当“取消”按钮被单击时取消当前操作。

对于在复选框、弹出式菜单和选项按钮中的鼠标单击，您的应用程序通常使用控制管理器例程 GetControlValue 和 SetControlVale 来获取并恰当地设定项目的值。本书“控制管理器”一章详细地解释了这些例程，但这章也给出了如何在您的警告栏和对话框中使用这些例程的实例。由于控制管理器并不知道选项按钮是如何分组的，所以当用户单击一个选项按钮时它并不自动关闭另一个选项按钮。这要由您的应用程序通过调用 GetControlValue 和 SetControlValue 例程来处理。

当用户单击“确定”按钮时，GetControlValue 针对每一个警告栏或对话框中显示的复选框和选项按钮返回相应的值，您的应用程序则应根据这些返回的值执行所有必要的动作。

当 ModalDialog 和 DiaogSelectTrack 调用 TrackControl 时，它们不允许您为比按钮、选项按钮或复选框更复杂的任何东西指定必要的特殊动作过程。如果您需要一个更复杂的控制部件（例如，测量用户已将鼠标按钮按下多长时间或已将一个指示符移动多远的控制部件），您可以在对话框中创建自己的控制部件（或图片，或者是绘制类似于控制部件的对象的定义应用程序的项目）。如果您使用 ModalDialog 过程，您就必须提供一个能够恰当处理该项目内事件的事件过滤函数；如果您使用 DialogSelect 函数，您就必须自己检测并响应这些事件。另一方面，您可以使用窗口管理器例程来显示合适的窗口，然后使用窗口管理器自己创建和管理这样复杂的控制部件。请参阅本书的“窗口管理器”和“控制管理器”两章，以便了解更多信息。

响应可编辑文字项目中的事件

当用户在您的对话框的可编辑文字项目中输入或编辑文字时，对话管理器调用TextEdit自动处理事件。(通常，您不应在警告栏中包括可编辑文字项目。)您一般总是停用可编辑文字项目，因为您通常并不需要知道用户每次是键入了一个字符还是单击了一个项目。只有当用户单击“确定”按钮时，才需要您确定可编辑文字项目中的文字。如6 - 49页的程序清单6 - 12所示，在用户单击了“确定”按钮之后，可以使用GetDialogItemText来确定可编辑文字项目的最终值。

在使用ModalDialog过程来处理定模式对话框中的事件时，并在将Dialog-Select函数用于不定模式或可移动定模式对话框时，对话管理器调用TextEdit来处理可编辑文字项目内的单击键和鼠标动作，使得:

- 当用户单击项目时，出现一个闪烁的 | 型光标指明可输入文字的插入点
- 当用户在项目中的文字上拖拽时，这些文字就被醒目显示；当用户双击一个词时，就突出显示这个单词；然后醒目显示的选定项就被用户键入的文字取代
- 如果在单击或拖拽的过程中用户按下Shift键，醒目显示的选定内容就相应地延长或缩短
- 当用户按下BackSpace键时，在插入点之前的醒目显示的选定项或字符就被删除
- 当用户按下Tab键时，光标将自动地进到项目到表资源的下一个可编辑文字项目，如果项目列表资源中已没有更多的项目就绕回到它的第一个项目

如果您的不定模式或可移动定模 Ω∂'ª∞ 框包含任何可编辑文字项目，就调用DialogSelect (甚至当WaitNextEvent返回FALSE时)。这是有必要的；因为；空事件期间DialogSelect函数调用TEIdle过程使文字光标在您的可编辑文字项目内闪烁；否则文字光标将不闪烁。程序清单6 - 25说明了应用程序定义的例程DoIdle，当应用程序的不定模式对话框是最前面窗口时，每当应用程序接收到空事件DoIdle就调用DialogSelect。

程序清单6 － 25 在空事件期间使用DialogSelect

```
PROCEDURE DoIdle (event: EventRecord);
VAR
    window:        WindowPtr;
    windowType:    Integer;
    itemHit:       Integer;
    result:        Boolean;
BEGIN
    window := FrontWindow;
    {determine which type of window--document, }
    { modeless dialog box, etc.--is in front}
    windowType := MyGetWindowType(window);
    CASE windowType OF
    kMyDocWindow: {document window is frontmost}
        ;    {see examples in "Event Manager" chapter}
    kMyGlobalChangesModelessDialog:  {modeless dialog is frontmost}
        result := DialogSelect(event, window, itemHit);
    END; {of CASE}
END;
```

一般说来，当您显示包含可编辑文字项目的对话框时，您的应用程序应该处理菜单栏存取。让您的编辑菜单处于有效状态，并使用DialogCut、DialogCopy、DialogPaste和DialogDelete过程来支持“剪”、“复制”、“贴”和“清除”命令以及这些命令的等效命令组合键。您也应该提供您自己的代码段支持“取消”命令。6－68页的“针对定模式对话框调整菜单”和6－73页的“针对可移动定模式和不定模式对话框调整菜单”这两节介绍了当您显示对话框时怎样允许用户存取您的编辑菜单。

如果您不提供自己的事件过滤函数且当一个定模式对话框在屏幕上时用户敲击Return或Enter键，对话管理器就不管对话框是否包含可编辑文字项目，都将把事件当作系统设定按钮(即列表中的第一个项目)上的鼠标单击对待。如果您的事件过滤函数对用户敲Return和Enter键作出响应，在可编辑文字项目中移动光标，那么就不要围绕任何按钮显示轮廓线。如果您的事件过滤函数就像响应用户单击系统设定按钮一样响应用户敲击Return和Enter，那么您就应该围绕系统设定按钮显示一个黑体轮廓线。请参阅6－86页的“为警告栏和定模式对话框编写事件过滤函数”一节，了解关于如何在您的对话框中把Return和Enter键映射到系统设定按钮的实例。

初时，一个可编辑文字项目可能包含系统设定文字，也可能不包含任何文字。您可以通过在项目列表资源中指定一个文字字符串作为这个项目的最后元素来提供系统设定文字，也可以使用SetDialogItemText过程（6－131页介绍了这个过程）来提供系统设定文字信息。

当首先显示一个包含可编辑文字项目的对话框时，插入点通常出现在项目列表资源的第一个可编辑文字项目中。而您可能想使用SelectDialogItemText过程，使得对话框出现时带有选定的文字，或使得用户输入文字出错时重新出现一个插入点或文字选定项。例如，当要求输入一个数值时，偶然地键入非数值输入的用户可以有重新输入的机会。6－ 131页中详细的介绍了SelectDialogItemText过程。

若为系统设定时，对话管理器用系统字体显示可编辑文字项目。如果希望保持各应用程序的视觉一致性，并使您的应用程序更容易实现本地化，那么您就不应该更改字体和字体大小。

响应警告栏中的事件

在显示警告栏或发出示警声之后，Alert，StopAlert，CautionAlert和NoteAlert函数将调用ModalDialog过程自动地处理事件。

ModalDialog过程又通过调用事件管理器函数GetNextEvent来获取每个事件。如果事件是警告栏内容区域之外的鼠标按下事件，ModalDialog就发出系统示警声并获取下一个事件。

Alert，StopAlert，CautionAlert和NoteAlert函数继续调用ModalDialog直至用户选定一个有效控制部件（一般为一个按钮）为止。这时这些控制部件就取消屏幕上的对话框并返回选定的控制部件的项目号码。然后，您的应用程序就对这个项目上的鼠标单击事件相应地作出响应。

例如，支持图6－39所示的警告栏的代码段必须响应3种不同的事件（用户可能单击的每一个按钮均需一个事件）。

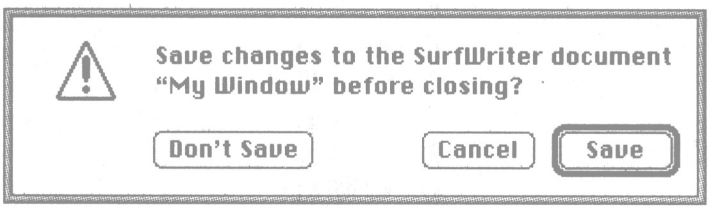


图6－39 CautionAlert为其报告事件的3个按钮

6－47页的程序清单6－9给出了一个应用程序定义的例程MyCloseDocument，用于“关闭”命令。如果自从上次存储文件以后又对其进行了修改，MyCloseDocument就在关闭窗口之前显示警告栏（图6－39说明了这个警告栏）。MyCloseDocument显示“当心”警告栏之后，它又检测它取消警告栏之后CautionAlent返回的项目号码。如果用户单击“存储”按钮，CautionAlert就返回它的项目号码，而MycloseDocument就调用其它应用程序定义的例程来存储文件、关闭文件并关闭窗口。如果用户单击“不存储”按钮，MyCloseDocument就关闭窗口但不存储文件。唯一可能的其它响应就是响应用户单击“取消”按钮，在这种情况下MyCloseDocument不做任何工作（对话管理器取消对话框，而MyCloseDocument只是让文稿窗口维持原状）。

标准事件过滤函数允许用户敲击Return或Enter键代替单击系统设定按钮。当敲击其中一个键时，标准事件过滤函数给ModalDialog返回TRUE，这样又使Alert、StopAlert、CautionAlert、和NoteAlert返回系统设定按钮的项目号码。当您编写您自己的事件过滤函数时，您的事件过滤函数应该仿效标准事件过滤函数，用这种方法响应涉及Return和Enter键的键盘事件。

对于警告栏内的事件，ModalDialog在处理事件之前把事件传递给一个事件过滤函数。事件过滤函数提供一个辅助事件处理循环，用来处理ModalDialog不处理的事件，用来超越ModalDialog要处理的事件。您应该为您的应用程序中的每个警告栏和定模式对话框提供一个简单的事件过滤函数。

您在Alert、StopAlert、CautionAlert和NoteAlert函数的第二个参数中指定您的事件过滤函数的指针。在6 - 47页所示的例程MycloseDocument中，指定了MyEventFilter函数的指针。大多数情况下，您可以在您的每个警告和定模式对话框中使用同样的事件过滤函数。6 - 86页的"为警告栏和定模式对话框编写事件过滤函数"一节中给出了一个简单的事件过滤函数实例，它允许后台应用程序接收更新事件并执行其它必要的事件处理。

除非您的事件过滤函数以它自己的方式处理事件并返回TRUE，否则Moda-lDialog 按以下方式处理警告栏内的事件：

- 在响应警告栏的激活或更新事件时，ModalDialog激活或更新其窗口
- 如果光标在一个控制部件中用户按下鼠标按钮，控制管理器函数TrackControl就跟踪鼠标。如果光标仍处在一个有效控制部件中时用户释放鼠标按钮，Alert，StopAlert，CautionAlert和NoteAlert，就取消警告栏并返回这个控制部件的项目号码（一般来说，按钮是您在警告栏中可以使用的唯一的控制部件）
- 如果光标在控制部件以外的任意一个有效项目中时用户按下鼠标按钮，Alert，StopAlert，CautionAlert和NoteAlert就取消警告栏并返回项目号码（一般来说，按钮控制部件应该是警告栏中的唯一有效项目）
- 如果光标在一无效项目中或不在任何项目中时用户按下鼠标按钮，或者如果发生任何其它的事件，Alert，StopAlert，CantionAlert和NoteAlert均不予理睬

响应定模式对话框中的事件

在显示定模式对话框之后立即调用ModalDialog过程。这个过程反复地处理定模式对话框内的事件，直到一个涉及有效项目的事件（如单击选项按钮）发生为止。如果事件发生在定模式对话框的内容区域之外，ModalDialog就发出系统示警声并获取下一个事件。在接收到一个涉及有效项目的事件之后，ModalDialog就返回项目号码。然后您就执行合适的动作响应项目中的事件。您的应用程序应该继续调用ModalDialog，直到用户选定"确定"或"取消"按钮为止，此时您的应用程序应该关闭对话框。

例如，如果用户单击一个选项按钮，您的应用程序就应该获取这个按钮的值，关掉其分组内其它任何选定的选项按钮，并重新调用ModalDialog来获取下一个事件。如果用户单击"取消"按钮，您的应用程序就应该将用户的工作还原至在用户调用对话框之前的状态，然后您的应用程序还应该在屏幕上取消对话框。

附注

不要把 ModalDialog 用于不定模式或可移动定模式对话框。◆

支持图 6 – 40 所示定模式对话框的代码段必须响应 4 个控制部件中的事件：两个复选框和两个按钮。

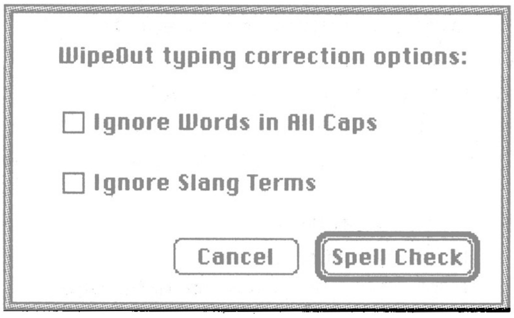


图 6 – 40 ModalDialog 报告其事件的 4 个项目

程序清单 6 – 26 给出了一个应用程序定义的例程 MySpellCheckDialog，它响应这 4 种控制部件中的事件。

程序清单 6 – 26 响应定模式对话框中的事件

```
FUNCTION MySpellCheckDialog: OSErr;
VAR
     docWindow:                     WindowPtr;
     ignoreCapsCheck:               Boolean;
     ignoreSlangCheck:              Boolean;
     spellDialog:                   DialogPtr;
     itemHit, itemType:             Integer;
     itemHandle:                    Handle;
     itemRect:                      Rect;
     capsVal:                       Integer;
     slangVal:                      Integer;
     event:                         EventRecord;
BEGIN
     capsVal := 0;
     slangVal := 0;
     ignoreCapsCheck := FALSE;
     ignoreSlangCheck := FALSE;
     MySpellCheckDialog := kSuccess; {assume success}
     docWindow := FrontWindow;       {get front window}
     IF docWindow <> NIL THEN
```

```
    DoActivate(docWindow, FALSE, event);{deactivate document window}
spellDialog := GetNewDialog(kSpellCheckID, NIL, Pointer(-1));
IF spellDialog = NIL THEN
BEGIN
    MySpellCheckDialog := kFailed;
    Exit(MySpellCheckDialog);
END;
MyAdjustMenus;                     {adjust menus as needed}
    GetDialogItem(spellDialog, kUserItem, itemType, itemHandle, itemRect);
    SetDialogItem(spellDialog, kUserItem, itemType,
              Handle(@MyDrawDefaultButtonOutline), itemRect);
    ShowWindow(spellDialog){show dialog box with default button outlined}
    REPEAT
        ModalDialog(@MyEventFilter, itemHit);
                                   {get events}
        IF itemHit = kAllCaps THEN {user clicked Ignore Words in All Caps}
        BEGIN
            {get the control handle to the checkbox}
            GetDialogItem(spellDialog, kAllCaps, itemType, itemHandle,
                          itemRect);
            {get the last value of the checkbox}
            capsVal := GetControlValue(ControlHandle(itemHandle));
            {toggle the value of the checkbox}
            capsVal := 1 - capsVal;
            {set the checkbox to the new value}
            SetControlValue(ControlHandle(itemHandle), capsVal);
        END;
        IF itemHit = kSlang THEN   {user clicked Ignore Slang Terms}
        BEGIN
        {get checkbox's handle, get its value, toggle it, then reset it}
        GetDialogItem(spellDialog, kSlang, itemType, itemHandle, itemRect);
            slangVal := GetControlValue(ControlHandle(itemHandle));
            slangVal := 1 - slangVal;
            SetControlValue(ControlHandle(itemHandle), slangVal);
        END;
    UNTIL ((itemHit = kSpellCheck) OR (itemHit = kCancel));
    DisposeDialog(spellDialog);    {close the dialog box}
    IF itemHit = kSpellCheck THEN  {user clicked Spell Check button}
    BEGIN
        IF capsVal = 1 THEN        {user wants to ignore all caps}
            ignoreCapsCheck := TRUE;
        IF slangVal = 1 THEN       {user wants to ignore slang}
            ignoreSlangCheck := TRUE;
```

```
        {now start the spell check}
        SpellCheckMyDoc(ignoreCapsCheck, ignoreSlangcheck);
    END;
END;
```

在使用GetNewDialog创建和显示对话框之后，MySpellCheckDialog例程应马上调用ModalDialog。MySpellCheckDialog例程不断地响应两个复选框中的事件，直到用户单击“拼写检查”或“取消”按钮为止。当用户单击其中一个复选框（在项目列表资源中的第三或第四个项目）时，MySpellCheckDialog使用GetDialogItem过程来获取这个复选框的间接指针。MySpellCheckDialog例程把这个间接指针强制转换成一个控制间接指针，并把它传递给控制管理器函数GetControlValue来获取控制部件的最后一个值（如果此复选框被选定则为1，如未选定则为0）。从这个值中减去1，MySpellCheckDialog得到这个控制部件的一个新的值。然后MySpellCheckDialog把这个值传递给控制管理器过程SetControlValue来设定这个新值。控制管理器作出响应，如果这个值为1就在框中绘制一个选择标记“×”，如果这个值为0则取消选择标记“×”。

如果用户单击“拼写检查”或“取消”按钮（它们是项目列表资源中的第一和第二个项目），MySpellCheckDialog就停止响应复选框中的事件。这个例程使用DisposeDialog过程（6－100页的“关闭对话框”一节中有关于这个过程的解释）来取消对话框。如果用户单击“取消”按钮，MySpellCheckDialog就不对对话框中的信息作进一步的处理。如果用户单击“拼写检查”按钮，MySpellCheckDialog就调用另一个应用程序定义的例程SpellCheckMyDoc，根据用户在复选框中给出的预置项，来检查文稿的拼写错误。

对于对话框内的事件，ModalDialog在处理事件之前把事件传递给事件过滤函数。在这个实例中，应用程序指定了它自己的事件过滤函数MyEventFilter的指针。您的应用程序应该提供一个事件过滤函数，在下一节中将对此作介绍。您可在您的大部分或所有的警告和对话框中使用同样的事件过滤函数。

除非您的事件过滤函数处理事件并返回TRUE，否则ModalDialog将按以下方式处理事件：

- 在响应对话框的激活或更新事件时，ModalDialog激活或更新其窗口
- 如果光标在一个可编辑文字项目中时用户按下鼠标按钮，ModalDialoy就相应地响应鼠标活动（即，要么显示插入点，要么选定文字）。如果发生一个键按下事件并且还有可编辑文字项目，就处理文字输入和编辑工作，从6－79页开始的“响应可编辑文字项目中的事件”一节对此作了介绍。如果可编辑文字项目是有效的，ModalDialog就在它 收按下鼠标或键按下事件之后返回项目号码。通常可编辑文字项目是无效的，您只有在用户单击“确定”按钮之后使用GetDialogItemText过程来读项目内的信息。6－49页的程序清单6－12说明了这种技术

■ 如果光标在控制部件中时用户按下鼠标按钮，ModalDialog就调用控制管理器函数 TrackControl。如果光标在一有效控制部件中时用户释放鼠标按钮，ModalDialog就返回控制部件的项目号码。您的应用程序应该相应地作出响应。例如，程序清单6 - 26使用一个应用程序定义的例程，当用户单击“拼写检查”按钮时它检查文稿的拼写

■ 如果光标在对话框内的任何其它有效项目中时用户按下鼠标按钮，ModalDialog就返回项目的号码，您的应用程序就作出相应的响应。一般来说，只有按钮控制部件是有效的。如果您的应用程序创建一个复杂的控制部件（如测量一个 Tick 盘移动了多远的控制功能)，您应用程序就必须提供事件过滤函数处理这个项目中的鼠标事件

■ 如果光标在一无效项目中或不在任何项目中时用户按下鼠标按钮，或者如果发生任何其它的事件，ModalDialog就不予理睬

为警告栏和定模式对话框编写事件过滤函数

对于警告栏和定模式对话框，对话管理器提供一个标准的事件过滤函数，检查用户是否敲击Enter或Return键，如果为真，就返回系统设定按钮的项目号码。在Macintosh系统软件的早期版本中，单任务环境中，用于警告和大多数定模式对话框的标准事件过滤函数一般就足够了。然而，由于标准的事件过滤函数并不允许后台应用程序接收或响应更新事件，所以它已经不能充分满足需要。

这样，您的应用程序就应该提供一个简单的事件过滤函数来执行这些功能，而且还应允许后台窗口接收更新事件。您可以在您的大多数警告和定模式对话框中使用同样的事件过滤函数。

您还可以使用您的事件过滤函数来处理ModalDialog不处理的其它事件，如“Command”+“.”组合键按下事件、磁盘插入事件、等效命令组合键以及您提供的定义应用程序的项目中的鼠标按下事件（如果必要)。

例如，标准的事件过滤函数忽略“Command”键的键按下事件。在您显示一个对话框之后，当您的应用程序允许用户存取您的菜单时，您的事件过滤函数应该处理菜单命令的等效命令组合键并返回TRUE。

至少，您的应用程序应该执行下列任务：

■ 如果用户敲击Return或Enter键，就返回 TRUE 和系统设定按钮的项目号码
■ 如果用户敲Esc键，或按下“Command”+“.”组合键，就返回TRUE和“取消”按钮的项目号码
■ 响应更新事件（这也允许后台应用程序接收更新事件）更新您的窗口并返回TALSE
■ 对于您的事件过滤函数不处理的所有事件，均返回FALSE

您也可以使用事件过滤函数检测并处理等效命令组合键和更复杂的事件(如用户在一定义应用程序的项目中拖拽光标)。例如，如果您提供一个定义应用程序的项目，它要求您测量用户按下鼠标按钮多长时间或用户拖拽光标多远，就使用事件过滤函数处理那个项目内的事件。

如果在这个事件过滤函数中您似乎将花费大量时间重复您的主要事件循环的很多工作，您就可以考虑在您的主要事件循环中处理所有的事件，而不使用对话管理器的 Alert、NoteAlert、StopAlert 和 CantionAlert 函数或 ModalDialog 过程。

您自己的事件过滤函数应该有 3 个参数并返回一个布尔值。例如，下面就是宣布一个叫做 MyEventFilter 的事件过滤函数的方法介绍:

```
FUNCTION MyEvertFilter£®theDialog£∫DialogPtr;
                     VAR theEvent£∫EventRecord;
                     VAR ItemHit£∫Integer£©£∫Boolean£ª
```

在接收到一个该函数不处理的事件之后，您的函数应该返回 FALSE。当您的函数返回 FALSE 时，ModalDialog 就处理您在参数 theEvent 中传递的这个事件。(您的应用程序也可以更改事件，冒充一个不同的事件并返回 FALSE，把改变了的事件传递给对话管理器处理。) 如果您的函数处理事件，您的函数就应该在 itemHit 参中返回 TRUE 作为一个函数结果，并返回它处理的项目的号码。而 ModalDialog 过程，Alert，NoteAlert，StopAlert 和 CautionAlert 函数在它们自己的 itemHit 参数中返回这个项目号码。

由于 ModalDialog 调用 GetNextEvent 函数时带有一个拒绝磁盘插入事件的事件类型屏蔽参数，您的应用程序就可调用事件管理器过程 SetSystemEventMask 来接受磁盘插入事件。请参阅本书“事件管理器一章”，了解关于处理磁盘插入事件的说明。

对于警告栏和定模式对话框，对话管理器提供一个标准的事件过滤函数，它检查用户是否敲击 Enter 或 Return 键，如果为真，就返回系统设定按钮的项目号码。您的事件过滤函数应该始终负责检查 Return 键或 Enter 键是否被按下，如果被按下，就在 itemHit 参数中返回系统设定按钮的项目号码，并作为一个函数结果返回 TRUE。您的事件过滤函数也应该检查是否按下 Esc 键，如果为真，在 itemHit 参数中返回“取消”按钮的项目号码，并返回一个函数结果 TRUE。您的应用程序还应该像响应用户单击“取消”按钮一样响应“Command” + “.” 组合键按下事件。

若要给出表明所选定项目的直观反馈，对于所有的警告和对话框，您应该反色显示等效命令组合键激活的按钮。首要原则是反色显示按钮 8 个 Tick 时间 ，这段时间足够引起用户的注意，但又不致太长使人觉得厌烦。每当用户单击按钮时控制管理器都执行这个动作，每当用户敲单击按钮的等效命令组合键时您的应用程序也应这样做。

对于包含可编辑文字项目的定模式对话框，您的应用程序应该处理菜单栏存取，以便允许用户使用您的编辑菜单及其“剪”、“复制”、“贴”、“清除”和“取消”命令，6－68 页的“为定模式对话框调整菜单”一节对此作了解释。然后，您的事件过滤函数应该检测并处理菜单栏中的鼠标按下事件和编辑菜单命令的等效组合键按下事件。您的应用程序应该通过使用过程 DialogCut、DialogCopy、DialogPaste 和 DialogDelete 来支持“剪”、“复制”、“贴”和“清除”命令，对用户选择的编辑菜单命令作出响应。

程序清单6 - 27给出了例程MyEventFilter，它从处理警告栏或对话框以外的窗口内的更新事件开始。(通过用这种方法响应您的应用程序自己的后台窗口的更新事件，您允许ModalDialog必要时执行一个次切换使得后台程序也能更新它们的窗口。)

下一步，MyEventFiliter处理激活事件。然后这个事件过滤函数就像处理用户单击系统设定按钮一样处理Return和Enter键的键按下事件，就像处理用户单击“取消”按钮一样处理Esc键的键按下事件。[请参阅《Inside Macintosh文本》*(Inside Macintosh:Text)*一书了解关于Return，Enter和Esc键的字符代码的信息。] 然后，您的事件过滤函数可以检测其它事件，如磁盘插入事件和等效命令组合键。

程序清单6 - 27 一个典型的用于警告和定模式对话框的事件过滤函数

```
FUNCTION MyEventFilter(theDialog: DialogPtr;
                       VAR theEvent: EventRecord;
                       VAR itemHit: Integer): Boolean;
VAR
     key:                      Char;
     itemType:                 Integer;
     itemHandle:               Handle;
     itemRect:                 Rect;
     finalTicks:               LongInt;
BEGIN
     MyEventFilter := FALSE;        {assume Dialog Mgr will handle it}
     IF (theEvent.what = updateEvt) AND
          (WindowPtr(theEvent.message) <> theDialog) THEN
          DoUpdate(WindowPtr(theEvent.message))
          {update the window behind}
     ELSE IF (theEvent.what = activateEvt) AND (WindowPtr(theEvent.message)
                    <> theDialog) THEN
          DoActivate(WindowPtr(theEvent.message),
               (BAnd(theEvent.modifiers, activeFlag) <> 0), theEvent)
     ELSE
          CASE theEvent.what OF
               keyDown, autoKey: {user pressed a key}
               BEGIN
                    key := Char(BAnd(theEvent.message, charCodeMask));
                    IF (key = Char(kReturnKey)) OR (key = Char(kEnterKey))
THEN
```

```
      BEGIN    {respond as if user clicked Spell Check}
          GetDialogItem(theDialog, kSpellCheck, itemType, itemHandle,
                              itemRect);
              {invert the Spell Check button for user feedback}
              HiliteControl(ControlHandle(itemHandle), inButton);
              Delay(kVisualDelay, finalTicks);
              {invert button for 8 ticks}
              HiliteControl(ControlHandle(itemHandle), 0);
              myEventFilter := TRUE;     {event's being handled}
              itemHit := kSpellCheck;
              {return the default button}
          END;
          IF (key = Char(kEscapeKey)) OR      {user pressed Esc key}
              (Boolean(BAnd(theEvent.modifiers, cmdKey)) AND
              (key = Char(kPeriodKey))) THEN  {user pressed Cmd-pd}
      BEGIN    {handle as if user clicked Cancel}
              GetDialogItem(theDialog, kCancel, itemType, itemHandle,
                             itemRect);
              {invert the Cancel button for user feedback}
              HiliteControl(ControlHandle(itemHandle), inButton);
              Delay(kVisualDelay, finalTicks);
              {invert button for 8 ticks}
              HiliteControl(ControlHandle(itemHandle), 0);
              MyEventFilter := TRUE;
              {event's being handled}
              itemHit := kCancel;
              {return the Cancel button}
          END;     {of Cancel}
          {handle any other keyboard equivalents here}
      END;     {of keydown, autokey}
      {handle disk-inserted and other events here, as needed}
      OTHERWISE
  END;     {of CASE}
END;
```

若要将这个事件过滤函数用于警告栏，当应用程序调用一个 Alert 函数时它指定 MyEventFilter 的指针，如 6 - 66 页的程序清单 6 - 19 所示。若要将这个事件过滤函数用于定模式对话框，当应用程序调用 ModalDialog 时它指定 MyEventFilter 的指针，如 6 - 83 页的程序清单 6 - 26 所示。

响应不定模式和可移动定模式对话框中的鼠标事件

若要处理不定模式和可移动定模式对话框中的事件，您可以使用IsDialogEvent函数来确定对话框中发生了事件。对于这样的事件，您可以使用DialogSelect函数自动地处理可编辑文字项目中的键按下事件，自动地处理更新和激活事件，并报告用户所单击的有效项目号码。您还必须使用辅助工具箱例程来处理其它类型的键盘事件和对话框中的其它事件。

警告

在System 7以前的系统软件版本中运行时，IsDialogEvent和DialogSelect函数不可靠。◆

另一方面，当您的应用程序响应不定模式和可移动定模式对话框中的事件时，可以首先确定发生的事件的类型，然后根据当前窗口的类型执行相应的动作，这大概是最有效的方法。如果一个不定模式或可移动定模式对话框是当前窗口，您就可以提供执行这个对话框的所有特定动作的代码段，然后使用DialogSelect函数代替控制管理器函数FindControl和TrackControl来处理您的对话框中的鼠标事件。DialogSelect函数也处理更新事件、激活事件和可编辑文字项目中的事件。(如果您的不定模式或可移动定模式对话包含可编辑文字项目，您就应该在空事件期间调用DialogSelect，使文字光标闪烁。)

如果您选择不借助IsDialogEvent函数来确定事件是否涉及可移动定模式或不定模式对话框，您的应用程序就应该作好准备处理下列鼠标事件:

- 鼠标单击菜单栏，您的应用程序已针对对话框对该菜单作了相应调整。在您的对话框的可编辑文字项目中，一定要使用DialogCut、DialogCopy、DialogPaste和DialogDelete过程支持“剪”、“复制”、“贴”和“清除”命令
- 当前可移动定模式或不定模式对话框内容区域中的鼠标单击事件。您可以使用DialogSelect 函数来帮助您处理事件
- 后台不定模式对话框内容区域中的鼠标单击事件。在这种情况下，您的应用程序应该把这个不定模式对话框变成当前活动的窗口
- 当可移动定模式或不定模式对话框为当前活动窗口时，一个后台窗口的内容区域中的鼠标单击事件。您的应用程序此时就应该发出系统示警声。而如果不定模式对话框，您的应用程序就应该把这个后台窗口送到前面
- 当前可移动定模式或不定模式对话框的拖拽区域(即标题栏)中的鼠标按下事件。您的应用程序应该使用窗口管理器过程DragWindow移动对话框来响应用户的动作
- 当一个可移动定模式对话框为当前窗口时，后台窗口拖拽区域中的鼠标按下事件。您的应用程序不应该移动这个后台窗口来响应用户的动作。相反，您的应用程序应该发出系统示警声
- 不定模式对话框的关闭格中的鼠标单击事件。您的应用程序应该处理或隐藏对话框，或执行其它更合适的动作

图 6 – 41 显示了一个简单的含有可编辑文字项目的不定模式对话框。

程序清单 6 – 28 说明了一个应用程序定义的过程，该过程可以处理所有窗口中的鼠标按下事件，包括图 6 – 41 所示的不定模式对话框。



图 6 – 41 DialogSelect 为其报告事件的一个不定模式对话框

程序清单 6 – 28 处理所有窗口的鼠标按下事件

```
PROCEDURE DoMouseDown (event: EventRecord);
VAR
     part:                    Integer;
     thisWindow:              WindowPtr;
BEGIN
     {find general location of the cursor at the time of mouse-down event}
     part := FindWindow(event.where, thisWindow);
     CASE part OF {take action based on the cursor location}
     inMenuBar: ;  {cursor in menu bar; respond with Menu Manager routines}
     inSysWindow: ;{cursor in a DA; use SystemClick here}
     inContent:    {cursor in the content area of one of this app's windows}
          IF thisWindow <> FrontWindow THEN
          BEGIN    {mouse-down in a window other than the front }
                   { window--make the clicked window the front window, }
                   { unless the front window is a movable modal dialog box}
               IF MyIsMovableModal(FrontWindow) THEN
                    SysBeep(30)       {emit system alert sound}
               ELSE
                    SelectWindow(thisWindow);
          END
          ELSE     {mouse-down in the content area of front window}
               DoContentClick(thisWindow, event);
     inDrag:       {handle mouse-down in drag area}
          IF (thisWindow <> FrontWindow) AND (MyIsMovableModal(FrontWindow))
          THEN
               SysBeep(30)   {emit system alert sound}
          ELSE

          DragWindow(thisWindow, event.where, GetGrayRgn^^.rgnBBox);
     inGrow: ;                    {handle mouse-down in zoom box here}
```

```
    inGoAway:      {handle mouse-down in close box here}
        IF TrackGoAway(thisWindow, event.where) THEN
            DoCloseCmd;
    inZoomIn, inZoomOut: ;
        {handle zoom box region for standard windows}
    END;           {end of CASE}
END;               {of DoMouseDown}
```

DoMouseDown 例程首先使用窗口管理器函数 FindWindow 大致确定鼠标按下时光标的所在位置。如果光标在一个窗口的内容区域，DoMouseDown 就通过比较 FindowWindow返回的窗口指针与窗口管理器函数FrontWindow返回的窗口指针，首先检查鼠标按下事件是否发生在当当前活动窗口中。

当鼠标按下事件发生在一个后台窗口时，DoMouseDown使用另一个应用程序定义的例程 MyIsMovableModal 来检查当前活动窗口是否是可移动定模式对话框。如果是，DoMouseDown 就发出系统示警声。否则，DoMouseDown 就使用窗口管理器过程 SelectWindow 使选定的窗口成为当前活动窗口。(MyIsMovableModal 例程使用窗口管理器函数GetWVariant来确定当前窗口的变量代码是否是MovableDBoxProc。如果是，MyIsMovableModal 就返回 TRUE。本书并未对此加以说明。）请参阅本书“窗口管理器”一章了解关于 SelectWindow 和 GetWVariant 例程更多的信息。

就像在这个实例中一样，您必须保证可移动对话框在您的应用程序范围内是定模式的。即当可移动定模式对话框是当前窗口时，用户不应该切换到您的应用程序的另一个窗口。相反，您的应用程序应该发出系统示警声。同时注意，当鼠标按下事件发生在任何窗口的拖拽区域时，DoMouseDown检查当一个可移动定模式对话框是当前窗口时该拖拽区是否属于一个后台窗口。如果是，DoMouseDown 又发出系统示警声。(但是，当您显示一个可移动定模式对话框时，通过在应用程序和苹果菜单中点按其它应用程序的窗口或选定其它应用程序，用户应该能够把您的应用程序切换到后台——当显示固定位置定模式对话框时用户不能执行这个动作。)

如果光标在当前活动窗口的内容区域时用户按下鼠标按钮，DoMouseDown就调用另一个应用程序定义的例程 DoContentClick 来进一步处理鼠标事件。程序清单6－29说明了在应用程序确定鼠标按下事件发生在6－91页的图6－41所示的不定模式对话框中之后，这个例程又是如何使 DialogSelect 函数来处理鼠标按下事件的。

程序清单 6 － 29 使用 DialogSelect 函数响应鼠标按下事件

```
PROCEDURE DoContentClick (thisWindow: windowPtr; event: EventRecord);
VAR
    itemHit:       Integer;
    refCon:        Integer;
BEGIN
    windowType := MyGetWindowType(thisWindow);
    CASE windowType OF
        kMyDocWindow: ;
    {handle clicks in document window here; see the chapter "Control }
            { Manager" for sample code for this case}
```

```
        kGlobalChangesID:      {user clicked Global Changes dialog box}
          BEGIN
             IF DialogSelect(event, DialogPtr(thisWindow), itemHit) THEN
                  BEGIN
                       IF itemHit = kChange THEN
             {user clicked Change}
                  ;    {use GetDialogItem and GetDialogItemText to get }
                       { the text strings and replace one string with the }
                       { other here}
                       IF itemHit = kStop THEN
                       {user clicked Stop}
                  ;    {stop making changes here}
                  END;
             END; {of CASE for kGlobalChangesID}
        {handle other window types here}
    END;     {of CASE}
END;
```

在这个实例中，当用户单击“更改”按钮时，DialogSelect返回它的项目号码。接下来，应用程序在用户的文稿范围之内执行“全局搜索”和“替换”。(6－49页的程序清单6－12介绍了将GetDialogItem和GetDialogItemText过程用于这种用途的方法。)一般来说，只有控制部件才应该在对话框中有效；因而，一般只有当用户单击一个有效控制部件之后DialogSelect返回TRUE时，您的应用程序才作出响应。例如，如果事件是对话框的激活或更新事件，DialogSelect就激活或更新对话框，所以您的应用程序并不需要响应这个事件。

这时，您可能也要核查并响应那些您不希望传递给DialogSelect或在传递给DialogSelect之前需要特殊处理的特殊事件。例如，如果对话框需要响应磁盘插入事件，那么这样做就很有必要了。

重要告诫

当DialogSelect调用TrackControl时，它不允许您指定更复杂控制部件必要的任何动作过程。例如，测量用户已按下鼠标按钮多长时间或测量用户已移动一个指示符多远的控制部件。对于这种情形，您可以创建一个图片或一个应用程序定义的绘制类似控制部件的对象的项目；然后您必须在把事件传递给DialogSelect之前核查这些事件并自己响应它们。或者，您可以使用控制管理器函数FindControl和TrackControl来处理您的对话框的控制部件内的鼠标事件。◆

6－91页的程序清单6－28每当用户单击当前活动窗口的关闭格时调用一个它的应用程序定义的例程DoCloseCmd。如果当前活动窗口是不定模式对话框，您会发现隐藏窗口会比取消它的数据结构更有效。程序清单6－30介绍了当用户单击找寻／替换不定模式对话框的关闭格时，您如何使用窗口管理器例程HideWindow来隐藏这个对话框。待用户再选择找寻／替换命令时，这个对话框已能使用，其状态与用户最近一次使用时相同，处于同样的位置，所选定的文字也一样。(6－67页的程序清单6－20介绍了首先怎样创建不定模式对话框，随后如何重新显示这个对话框。)

程序清单 6 – 30 响应关闭命令隐藏不定模式对话框

```
PROCEDURE DoCloseCmd;
VAR
     myWindow: WindowPtr;
     myData: MyDocRecHnd;
     windowType: Integer;
BEGIN
     myWindow := FrontWindow;
     windowType := MyGetWindowType(myWindow);
     CASE windowType OF
          kMyGlobalChangesModelessDialog:
               HideWindow(myWindow);
          kMySpellModelessDialog:
               HideWindow(myWindow);
          kMyDocWindow:
               BEGIN
                    myData := MyDocRecHnd(GetWRefCon(myWindow));
                    MyCloseDocument(myData);
               END;      {of kMyDocWindow case}
          kDAWindow:
               CloseDeskAcc(WindowPeek(myWindow)^.windowKind);
     END; {of CASE}
END;
```

响应不定模式和可移动定模式对话框中的键盘事件

如果您采纳前面介绍的确定（不借助 IsDialogEvent 函数）事件是否涉及可移动或不定模式对话框的策略，您的应用程序应该准备处理下列键盘事件：

- 等效命令组合键。如用来拷贝的“Command” + “C”，您的应用程序应该对它作出合适的响应
- Return 和 Enter 键的键按下事件。您的应用程序应该像响应用户单击系统设定按钮一样对它们作出响应
- Esc 或“Command” + “.”键按下事件。您的应用程序应该像响应用户单击“取消”按钮一样对它们作出响应
- 可编辑文字项目中的键按下和自动键事件。您的应用程序可针对它们使用 DialogSelect 函数，这个函数又调用 TextEdit 来自动地处理可编辑文字项目内的按键事件

程序清单6－31说明了每当一个应用程序接收键按下事件时它是如何核查等效命令组合键的。如果用户按下Command键的同时单击另一个键，应用程序就调用它的另一个应用程序定义的过程DoMenuCommand，它处理菜单命令的等效命令组合键。请参阅本书“菜单管理器”一章了解DoMenuCommand过程的一个实例。记住当可移动定模式或不定模式对话框是现用时，您的应用程序应该相应地调整菜单，并使用DialogCut、DialogCopy、DialogPaste和DialogDelete过程在可编辑文字项目中支持“剪”、“复制”、“贴”和“清除”命令。

程序清单6－31 核查涉及Command键的键按下事件

```
PROCEDURE DoKeyDown (event: EventRecord);
VAR
     key:      Char;
BEGIN
     key := CHR(BAnd(event.message, charCodeMask));
     IF BAnd(event.modifiers, cmdKey) <> 0 THEN
     BEGIN         {Command key down}
          IF event.what = keyDown THEN
          BEGIN
               MyAdjustMenus;         {adjust the menus as needed}
               DoMenuCommand(MenuKey(key));
               {handle the menu command}
          END;
     END
     ELSE
          MyHandleKeyDown(event);
END;
```

在确定一个键按下事件不涉及等效命令组合键之后，程序清单6－31调用它自己的另一个例程MyHandleKeyDown，程序清单6－32给出了这个例程。

列表 6 － 32 核查不定模式对话框中的键按下事件

```
PROCEDURE MyHandleKeyDown (event: EventRecord);
VAR
     window:             WindowPtr;
     windowType:         Integer;
BEGIN
     window := FrontWindow;
     {determine the type of window--document, modeless, etc.}


     windowType := MyGetWindowType(window);
     IF windowType = kMyDocWindow THEN{key-down in doc window}
     BEGIN          {handle keystrokes in document window here}
     END
     ELSE           {key-down in modeless dialog box}
          MyHandleKeyDownInModeless(event, windowType);
END;
```

MyHandleKeyDown 确定当用户单击键时当前活动窗口的类型。如果不定模式对话框是最前面窗口，MyHandleKeyDown 就自动地调用另一个应用程序定义的例程 MyHandleKeyDownInModeless 来响应不定模式对话框中的键按下事件。程序清单 6 － 33 给出了 MyHandleKeyDownInModeless 例程。

程序清单 6 － 33 响应不定模式对话框中的键按下事件

```
PROCEDURE MyHandleKeyDownInModeless(event: EventRecord; windowType:
          Integer);
VAR
     key:                          Char;
     itemType:                     Integer;
     itemHandle:                   Handle;
     itemRect:                     Rect;
     finalTicks:                   LongInt;
     handled:                      Boolean;
     item:                         Integer;
     theDialog:                    DialogPtr;
BEGIN
     handled := FALSE;
     theDialog := FrontWindow;
     CASE windowType OF
          kGlobalChangesID: {key-down in Global Changes dialog box}
          BEGIN
               key := Char(BAnd(event.message, charCodeMask));
               IF (key = Char(kReturnKey)) OR (key = Char(kEnterKey)) THEN
               BEGIN          {respond as if user clicked Change}
                    GetDialogItem(theDialog, kChange, itemType, itemHandle,
                                   itemRect);
```

```
            {invert the Change button for 8 ticks for user feedback}
            HiliteControl(ControlHandle(itemHandle), inButton);
            Delay(kVisualDelay, finalTicks);
            HiliteControl(ControlHandle(itemHandle), 0);
        {use GetDialogItem and GetDialogItemText to get the text }
        { strings and replace one string with the other here}
            handled := TRUE;        {event's been handled}
        END;
            IF (key = Char(kEscapeKey)) OR {user pressed Esc key}
                (Boolean(BAnd(event.modifiers, cmdKey)) AND
                (key = Char(kPeriodKey))) THEN{user typed Cmd-pd}
                BEGIN    {handle as if user clicked Stop}
            GetDialogItem(theDialog, kStop, itemType, itemHandle,
                                        itemRect);
            {invert the Stop button for 8 ticks for user feedback}
            HiliteControl(ControlHandle(itemHandle), inButton);
                    Delay(kVisualDelay, finalTicks);
                    HiliteControl(ControlHandle(itemHandle), 0);
                        {cancel the current operation here}
                    handled := TRUE;  {event's been handled}
                END;
IF NOT handled THEN{let DialogSelect handle keydown events in }
                                { editable text items}
                    handled := DialogSelect(event, theDialog, item);
            END;     {of case kGlobalChangesID}
        {handle other modeless and movable modal dialog boxes here}
        END;     {of CASE}
END;
```

当MyHandleKeyDownInModeless确定前面窗口是找寻/替换不定模式对话框时，它检查用户是否单击了Return或Enter键。如果单击了，MyHandKeyDownInModeless就像用户已单击了系统设定按钮Change一样来响应。MyHandleKeyDownInModeless例程使用控制管理器过程HiliteControl来醒目显示Change按钮8个Tick时间。(6－88页的程序清单6－27说明了如何在定模式对话框的事件过滤函数中使用HiliteControl来醒目显示按钮。)

当用户敲击Esc或"Command"＋"."组合键，MyHandleKeyDownInModeless就像用户已单击了"取消"按钮一样来响应。

最后，MyHandleKeyDownInModeless使用DialogSelect函数，这个函数又调用TextEdit来处理可编辑文字项目内的单击键事件。

响应不定模式和可移动定模式对话框中的激活和更新事件

如果您采纳了前面介绍的确定(不借助IsDialogEvent函数)事件是否涉及可移动或不定模式对话框的策略，您的应用程序就应该准备处理可移动定模式和不定模式对

话框的激活和更新事件。您可使用DialogSelect来帮助您处理激活和更新事件。为了使得速度更快，在处理更新事件时可能要借助UpdateDialog函数。DialogSelect和UpdateDialog都使用QuickDraw过程SetPort，在刷新或更新对话框之前，使对话框成为当前图形端口。

当您停用一个不定模式或可移动定模式对话框时，您应该使用控制管理器过程HiliteControl使其中的按钮和其它控制部件变成无效。HiliteControl过程使按钮、选项按钮、复选框和弹出式菜单的彩色变暗淡，向用户表明当对话框在后台时单击这些项目不起任何作用。当您重新激活不定模式或可移动定模式对话框时，您应该使用HiliteControl使控制部件重新变成有效。

程序清单6－34给出了应用程序定义的例程DoActivateGlobalChangesDialog，它说明了当激活不定模式对话框时怎样使用HiliteControl使Change按钮变成有效，当停用对话框时又如何使Change和Stop按钮变成无效。

程序清单6－34 激活不定模式对话框

```
PROCEDURE DoActivateGlobalChangesDialog (window: WindowPtr;
                                         event: EventRecord);
VAR
     activate:                    Boolean;
     handled:                     Boolean;
     item:                        Integer;
     itemType:                    Integer;
     itemHandle:                  Handle;
     itemRect:                    Rect;
BEGIN
     MyCheckEvent(event);{get a valid event record to pass to DialogSelect}
     activate := (BAnd(event.modifiers, activeFlag) <> 0);
     IF activate THEN       {activate the modeless dialog box}
     BEGIN
          {highlight editable text}
          SelectDialogItemText(window, kFindText, 0, 32767);
          {make the Change button active (make the Stop button active }
          { only during a change operation)}
          GetDialogItem(DialogPtr(window), kChange, itemType, itemHandle
          itemRect);
     HiliteControl(ControlHandle(itemHandle), 0); {make Change active}
          {draw a bold outline around the newly activated Change button}
          MyDrawDefaultButtonOutline(DialogPtr(window), kChange);
     END
     ELSE {dim the Change and Stop buttons for a deactivate dialog box}
     BEGIN
          GetDialogItem(DialogPtr(window), kChange, itemType, itemHandle,
                                 itemRect);
          HiliteControl(ControlHandle(itemHandle), 255);
          {dim Change button}
```

```
        {draw a gray outline around the newly dimmed Change button}
        MyDrawDefaultButtonOutline(DialogPtr(window), kChange);
        GetDialogItem(DialogPtr(window), kStop, itemType, itemHandle,
                                 itemRect);
        HiliteControl(ControlHandle(itemHandle), 255);
        {dim Stop button}
   END;
   {let Dialog Manager handle activate events}
   handled := DialogSelect(event, window, item);
   MyAdjustMenus;     {adjust the menus appropriately}
   END;
```

DoActivateGlobalChangesDialog例程使用DialogSelect来处理不定模式对话框中的激活事件。响应活跃事件时，DialogSelect处理事件并返回FALSE。每当用户使不定模式对话框变成现用时，DialogSelect函数就把不定模式对话框设定为当前图形端口。

由于DialogSelect需要3个参数，其中之一必须是一个事件记录，DoActivateGlobalChangesDialog使用应用程序定义的例程MyCheckEvent来验证事件是有效事件。如果不是，MyCheckEvent就创建并返回激活事件的有效事件记录。

因为DialogSelect响应活跃事件时没有针对项目调用任何绘图过程，所以DoActivateGlobalDialog调用应用程序定义的绘图例程MyDrawDefaultButtonOutline，在激活对话框时围绕系统设定按钮绘制黑色轮廓线，停用对话框时则绘制灰色轮廓线。6－59页的程序清单6－17给出了MyDrawDefaultButtonOutline例程。

因为当您显示一个可移动定模式对话框时，用户可以从您的应用程序中切换出来，所以您的应用程序也必须处理可移动定模式对话框的激活事件。

您也可使用DialogSelect来处理更新事件。响应更新事件时，DialogSelect调用窗口管理器过程BeginUpdate，调用对话管理器过程DrawDialog来刷新整个对话框，然后调用窗口管理器过程EndUpdate。然而，更新对话框的一个快捷方法是使用UpdateDialog过程，它只重新绘制对话框的更新区域。如程序清单6－35所示，您应该在使用UpdateDialog之前调用BeginUpdate，最后调用EndUpdate。

程序清单6－35 更新不定模式对话框

```
PROCEDURE DoUpdate (window: WindowPtr);
VAR
        windowType: Integer;
BEGIN
     windowType := MyGetWindowType(window);
     CASE windowType OF
          kMyDocWindow:
                    {update document windows here}
```

```
                kMyGlobalChangesModelessDialog:
                    BEGIN
                        BeginUpdate(window);
                        UpdateDialog(window, window^.visRgn);
                        EndUpdate(window);
                    END;
                {handle cases for other window types here}
            END; {of CASE}
        END;
```

关闭对话框

当您不再需要对话框时，如果您为对话框分配存储器，您可使用CloseDialog过程处理它；如果不为它分配存储器，就使用DisposeDialog过程；或者，您可通过使用窗口管理器过程HideWindow仅使对话框变成不可视。

一般说来，您的应用程序不应该为定模式对话框或可移动定模式对话框分配存储器，但它应该给不定模式对话框分配存储器。在这些情况下，当用户单击“确定”或“取消”按钮时您的应用程序应该使用DisposeDialog来处理固定或可移动定模式对话框；当用户单击关闭格或在文件菜单中选择关闭时您的应用程序应该使用CloseDialog来处理不定模式对话框。

您不用关闭警告栏。在用户单击任何有效按钮响应警告栏之后，对话管理器通过调用DisposeDialog过程自动地为您完成这项工作。

CloseDialog过程在屏幕上取消对话框并在窗口列表中删除它。CloseDialog还释放下列对象占据的存储器：

- 与对话框相关联的数据结构（如它的结构、存储器和更新区域）
- 对话框内的所有项目（图片和图标除外，它们可能被其它资源共享）和与它们相关联的所有数据结构（例如，滚动杆的滚动块所占据的区域）

CloseDialog过程不处理对话记录或项目列表资源。与GetNewDialog不同，NewDialog不使用项目列表资源的备份。这样，如果您用NewDialog创建一个对话框，您就可能要使用CloseDialog在存储器中保持项目列表资源（即使您没提供存储器的指针）。

DisposeDialog过程调用CloseDialog，此外还释放对话的项目列表资源和对话记录所占据的存储器。如果您给GetNewDialog或NewDialog传递参数NIL，让对话管理器在堆中分配存储器，当您完成对话框的工作时就调用DisposeDialog。

对于不定模式和可移动定模式对话框，您可能会发现隐藏对话框比取消它的数据结构效速率更高。当用户单击找寻／替换不定模式对话框的关闭格时，6－94页的程序清单6－30使用窗口管理器例程HideWindow来隐藏这个对话框。待下次用户调用找寻／替换命令时，对话框已能使用，其状态与用户最后使用它时相同，处于同样的位置，所选定的文字也相同。

如果您显示一个对话框时调整菜单，当您关闭这个对话框时一定要把菜单返回到合适的状态。6 － 68 页的“针对定模式对话框调整菜单”部分和 6 － 73 页的“针对可移动定模式对话框和不定模式对话框调整菜单”这部分都对此作了介绍。

对话管理器参考资料

这部分介绍对话管理器特定的数据结构、例程和资源。

“数据结构”这部分给出了对话管理器创建并保留的对话记录的 Pascal 数据结构。“对话管理器例程”这部分介绍了一些对话管理器例程，它们用于调用警告栏、创建和处理对话框、操纵警告栏和对话框中的项目和处理对话框中的事件。

当您在对话框中需要创建定义应用程序的项目、需要过滤对话管理器不处理的事件、需要定义自己的示警声时，您的应用程序必须供给自己的例程，“应用程序定义的例程”这部分就介绍了这些例程。

“资源”这部分介绍了对话资源、警告资源、项目列表资源、对话项目色彩表资源、警告彩色表资源和项目彩色表资源。本章的小结部分包括了下列常数的列表：定义警告栏和对话框中项目类型值的常数，定义警告栏中“确定”和“取消”按钮的值的常数，定义“注意警告栏”内图标的值的常数，定义“当心”和“停止”警告栏的值的常数，以及对话管理器的格式塔函数使用的常数。

数据结构

这部分介绍对话记录。您的应用程序没有必要创建或使用这个记录；相反，您的应用程序只需使用合适的对话管理器例程，创建所有必要的资源，然后允许对话管理器根据需要创建和使用这种数据类型的记录。这里仅介绍对话记录的完整性。

对话记录

若要创建警告栏和对话框，您使用对话管理器例程（如 Alert 或 GetNewDialog），这个例程把您的项目列表资源和您的警告资源（或对话资源）中的信息溶合到存储器中叫做“对话记录”的数据结构里。每当您的应用程序创建一个警告栏或对话框时，对话管理器就创建一个对话记录，它是 DialogRecord 类型的数据结构。您的应用程序一般不应该创建对话记录或直接存取它的域。

```
TYPE      DialogPtr                 = WindowPtr;
          DialogPeek                = ^DialogRecord
          DialogRecord        =
          RECORD
            window:     WindowRecord; {dialog window}
            items: Handle;        {item list resource}
            textH: TEHandle;      {current editable text item}
            editField:Integer;    {editable text item number }
                                  { minus 1}
            editOpen:Integer;     {used internally; reserved}
            aDefItem:Integer;     {default button item number}
          END;
```

域说明

window	警告栏或对话框的窗口记录。
items	警告或对话框的项目列表资源的间接指针。
textH	当前可编辑文字项目的间接指针。
editField	当前可编辑文字项目
editOpen	内部使用；保留。
aDefItem	系统设定按钮的项目号码

对话管理器例程

这部分介绍用来初始化对话管理器、调用警告栏、创建和处理对话框、操纵警告栏和对话框中的项目和处理警告和对话框中的事件的例程。

对于一些对话管理器例程,您可使用拼写不同的例程名称存取它们,这取决于您的开发环境所支持的界面文件。例如，存取GetDialogItem时也可用GetDItem。表6 - 1提供了例程以前的名称和它的新等效名称之间的映射。

表 6 － 1　对话管理器例程的新名称和以前的名称之间的映像

新名称	旧名称
DialogCopy	DlgCopy
DialogCut	DlgCut
DialogDelete	DlgDelete
DialogPaste	DlgPaste
DisposeDialog	DisposDialog
FindDialogItem	FindDItem

(续表6 - 1)

新名称	旧名称
GetAlertStage	GetAlrtStage
GetDialogItem	GetDItem
GetDialogItemText	GetIText
HideDialogItem	HideDItem
NewColorDialog	NewCDialog
ResetAlertStage	ResetAlrtStage
SelectDialogItemText	SelIText
SetDialogFont	SetDAFont
SetDialogItem	SetDItem
SetDialogItemText	SetIText
ShowDialogItem	ShowDItem
UpdateDialog	UpdtDialog

初始化对话管理器

在使用对话管理器之前，您必须按顺序初始化QuickDraw、字体管理器、窗口管理器、菜单管理器和TextEdit。第一个调用的对话管理器例程是InitDialogs过程，它初始化对话管理器。

根据您的应用程序的请求，对话管理器在各个警告阶段期间使用系统示警声向用户发出信号。对于警告，如果您想让对话管理器发出的语音不是系统示警声，就编制您自己的示警声过程（6 - 144页中有介绍)，并调用ErrorSound过程使它成为当前示警声过程。

系统设定时，对话管理器用系统字体显示静态文字和可编辑文字项目。为了在全世界范围内的各种系统软件版本中使用您的应用程序时更易实现本地化，您不应该更改字体。但是，如果您决定您的应用程序确实有必要用系统字体以外的字体来显示静态文字或可编辑文字，您就可使用 SetDialogFont 过程。

InitDialogs

使用 InitDialogs 过程初始化对话管理器。

```
PROCEDURE InitDialogs (resumeProc: ResumeProcPtr);
```

resumeProc 在MultiFinder以前的系统中发生严重系统错误时系统错误处理器使用的过程的指针。在 System 7 中，您的应用程序应该把这个参数设定为NIL。

在使用对话管理器之前，您必须按照这个顺序初始化QuickDraw、字体管理器、窗口管理器、菜单管理器和TextEdit。然后在调用所有其它对话管理器例程之前调用InitDialogs初始化对话管理器。InitDialogs过程执行下列初始化工作:

- 存储在resumeProc参数中传递的指针。在System 7中，您的应用程序应该把resumeProc参数设定成NIL
- 安装系统示警声。若要更改系统示警声，使用ErrorSound过程
- 向ParamText过程传递空字符串

ErrorSound

若要用您自己的示警声代替系统示警声向用户发信号，使用ErrorSound过程。

```
PROCEDURE ErrorSound (SoundProc: SoundProcPtr);
```

SoundProc　生成想要的示警声的过程的指针。

说明

对话管理器在各个警告阶段期间使用系统示警声向用户发信号。系统示警声是存储在系统文件中的语音资源，每当系统软件或您的应用程序使用语音管理器过程SysBeep时，就发出系统示警声。通过更改语音控制板的设定状态，用户可以确定发出哪种语音。如果您想在各个警告阶段使用系统示警声以外的语音，就编写您自己的示警声过程并调用ErrorSound过程使它成为当前示警声过程。

特殊考虑

如果您在SoundProc参数中传递NIL，对话管理器就既不发出语音也不使菜单闪烁，这样用户就接收不到任何信号。

亦请参阅

请参阅6－144页关于MyAlertSound的说明，了解关于如何编写SoundProc参数指向的示警声过程的讨论。若想了解如何将语音警告溶合到警告阶段的实例，请参阅6－21页的程序清单6－2和6－22页的程序清单6－3。

SetDialogFont

尽管您一般不更改静态和可编辑文字项目中使用的字体，但您仍可通过 SetDialogFont 过程来这样做。SetDAFont 过程与 SetDialogFont 过程是等同的。

```
PROCEDURE      SetDialogFont (FontNum: Integer);
```

FontNum　字体标识符号码。不要依赖于字体号码常数。而应使用字体管理器函数 GetFNum 来查找在此参数中传递的字体号码。

说明

对于随后创建的对话框和警告栏，SetDialogFont 将对话框或警告栏的图形端口的字体设定成指定的字体。如果您不调用这个过程，就使用系统字体。SetDialogFont 过程并不影响控制部件的标题，它始终用系统字体显示。

特殊考虑

关于 SetDialogFont 过程，需要注意以下几点。

首先，一旦调用这个过程，标准文件软件包就不能保证正确地计算标准文件对话框的位置。例如，标准文件对话框可能被菜单栏部分地遮住。其次，要意识到这个过程影响到其后显示的所有警告栏和对话框中的所有静态文字和可编辑文字项目。第三，SetDialogFont 并不更改控制标题。第四，您不能使用 SetDialogFont 来更改字体大小或字形。最后，也是最重要的一点，如果您从不使用 SetDialogFont，而是在您的警告栏和对话框中使用系统字体，您的应用程序就会很容易实现本地化。

亦请参阅

请参阅《Inside Macintosh 文本》*(Inside Macintosh: Text)* 一章，了解关于 GetFNum 函数的信息。

创建警告

若要创建警告（包括示警声、警告栏或以上两者），可以使用下列函数之一：NoteAlert，CautionAlert，StopAlert和Alert。前3个函数分别在警告栏的左上角显示“注意”、“当心”和“停止”警告图标（请参阅图6－3、图6－4和图6－5）。Alert函数允许您在对话框的左上角显示您自己的图标或者不显示任何图标。

这些函数从您提供的警告资源中获取关于警告的说明信息。当您调用它们之中的一个函数时，您向它传递警告资源的资源标识符和事件过滤函数的指针。这些函数将根据您在警告资源中规定的警告阶段建立对话记录、发出示警声和显示警告栏。

当您调用Alert、StopAlert、CautionAlert和NoteAlert函数时，您应该指定事件过滤函数的指针。您可以在大多数警告和定模式对话框中使用同样的事件过滤函数。

如果您需要了解当前警告阶段（例如，如果在某个阶段要显示警告栏就要保证您的应用程序停用最前面的窗口。），就使用GetAlertStage函数。如果需要更改当前警告阶段，可以使用ResetAlertStage过程。

您的应用程序不处理警告栏中的事件；对话管理器可以自动地为您完成这项工作。

Alert

若要显示警告栏（或者，如果与警告阶段相适应，发出系统示警声取代警告栏，或在显示警告栏之外又发出系统示警声），您可使用Alert函数。这个函数不在警告栏的左上角显示系统设定图标。您可使这个区域保持空白，或者在警告的项目列表资源中指定您自己的图标，在警告资源中又指定这个项目列表资源。

```
FUNCTION Alert (alertID: Integer;
                filterProc: Modal FilterProcPtr): Integer;
```

alertID　警告资源的资源标识符。如果警告资源丢失，对话管理器就不创建所请求的警告而返回到您的应用程序。

filterProc 响应ModalDialog过程不处理的事件的函数的指针。

说明

Alert函数创建在指定的警告资源中定义的警告。这个函数调用当前示警声过程并向它传递在当前警告阶段的警告资源中指定的语音号码。如果在这个阶段没有绘制警告栏，Alert就返回－1；否则，它就使用NewDialog函数来创建和显示警告栏。如果您的应用程序不提供与警告资源具有相同的资源标识符的警告彩色表资源，就使用系统设定系统彩色。

Alert函数使用ModalDialog过程，这个过程反复地为您获取和处理大部分事件。ModalDialog又通过调用事件管理器函数GetNextEvent来获取每个事件。如果事件是一个在警告栏的内容区域之外的鼠标按下事件，ModalDialog就发出错误语音并获取下一个事件。

Alert函数继续调用ModalDialog直到用户选定一个有效控制部件为止（一般是按钮），此时Alert函数在屏幕上取消警告栏并返回选定控制部件的项目号码。然后当用户点按这个项目时您的应用程序作出合适的响应。

对于警告栏内的事件，ModalDialog在处理它之前把它传递给事件过滤函数。事件过滤函数为ModalDialog不处理的事件提供一个辅助事件处理循环。您在Alert函数的filterProc参数中指定您的事件过滤函数的指针。

如果您将filterProc参数设定成NIL，对话管理器就使用标准的事件过滤函数，其表现如下:

- 如果用户敲击Return或Enter键，事件过滤函数就返回TRUE并返回系统设定按钮的项目号码

但是，您的应用程序应该提供一个简单的事件过滤函数，它不仅具有以上表现，而且还有下列表现:

- 如果用户敲击Esc或“Command”+“.”组合键，就返回TRUE和“取消”按钮的项目号码
- 响应更新事件时更新您的窗口（这也允许后台窗口接收更新事件）并返回FALSE
- 对于您的事件过滤函数不处理的所有事件均返回FALSE

您也可以使用事件过滤函数检测和响应等效命令组合键。

除非事件过滤函数用它自己的方法处理事件并返回TRUE，否则ModalDialog就会按如下方法处理警告栏内的事件。

- 如果光标在控制部件中时用户按下鼠标按钮，控制管理器函数TrackControl就跟踪光标。如果光标仍处在有效控制部件中时用户释放鼠标按钮，Alert，StopAlert，CautionAlert和NoteAlert就取消警告栏并返回控制部件的项目号码。（一般来说，按钮应是警告栏中的唯一一种控制部件。）
- 如果光标在控制部件以外的任何其它有效项目中时用户按下鼠标按钮，Alert，StopAlert，CautionAlert和NoteAlert就取消警告栏并返回项目号码。（一般说来，按钮控制部件应该是警告栏中仅有的有效项目。）
- 如果光标在一个无效的项目中、或不在任何项目中时用户按下鼠标按钮，或者如果发生任何其它的事件，Alert，StopAlert，CautionAlert和NoteAlert就不予理睬

Alert函数使用QuickDraw例程SetPor使警告栏成为当前图形端口。在显示警告栏之前您的应用程序没有必要再调用 SetPort, 因为在您创建警告栏到对话管理器显示它的这段时间内您不能在任何其它窗口内绘图。

特殊考虑

当您的应用程序跑在后台时，或者不可视时，如果您要显示警告栏，您就应该使用通知管理器向用户发送一个通知。通知管理器自动地显示包含您所指定的消息的警告栏；您将没有必要使用对话管理器自己创建对话框。

注意，通知管理器提供了从您的应用程序到用户的单向通信通道。当您的应用程序在后台时没有任何把信息从用户传返回应用程序的措施(尽管您的应用程序有可能确定通知是否被接收)。如果您需要征求用户的信息，就使用通知管理器通知用户把您的应用程序送到前台，在这里用户可以响应您的应用程序展示的警告栏。

亦请参阅

6 - 135页介绍了ModalDialog过程。请参阅6 - 86页的“为警告和定模式对话框编写事件过滤函数”这部分了解关于如何编写事件过滤函数的讨论。请参阅6 - 18页的“创建示警声和警告栏”部分了解关于警告和警告阶段的讨论。请参阅6 - 37页的“按钮、复选框和选项按钮的标题”部分和6 - 40页的“静态文本项目和可编辑文字项目的文字字符串”部分，了解关于按钮标题和警告栏内消息的建议。6 - 150页介绍了“警告资源”。6 - 157页介绍了“警告彩色表资源”。对话管理器使用系统示警声作为错误语音，除非您调用ErrorSound过程更改它，6 - 104页对此作了介绍。请参阅6 - 81页的“响应警告栏中的事件”部分了解关于如何响应Alert函数返回的事件的讨论。请参阅《Inside Macintosh 进程》*(Inside Macintosh: Process)* 的“通知管理器”一章了解关于通知管理器的信息。

NoteAlert (6 - 110页作了介绍)、CautionAlert (6 - 111页作了介绍) 和StopAlert (即将介绍) 都在警告栏的左上角显示图标，除此之外，它们与Alert函数完全一样。

StopAlert

若要显示一个左上角有"停止"图标的警告栏（或，如果与警告阶段相适应，发出示警声代替警告栏，或者除警告栏以外还发出示警声），使用 StopAlert 函数。

```
FUNCTION StopAlert (alertID: Integer;
                    filterProc: ModalFilterProcPtr): Integer;
```

alertID　警告资源的资源标识符。如果警告资源丢失，对话管理器就返回到您的应用程序，而不创建请求的警告。

filterProc 响应 ModalDialog 过程不处理的事件的函数的指针。如果您把这个参数设定成 NIL，对话管理器就使用标准的事件过滤函数，它允许用户敲击 Return 或 Enter 键代替单击系统设定按钮。但是，您的应用程序应该提供一个简单的也允许后台应用程序接收更新事件的事件过滤函数。在这个参数中传递事件过滤函数的指针。

说明

在绘制警告栏内的项目之前，StopAlert 先在左上角（在局部坐标为 [10, 20, 42, 52] 的矩形内）绘制"停止"图标，除此之外，StopAlert 函数与 Alert 函数是相同的。"停止"图形的资源标识符如下：

```
CONST StopIcon=0;  {stop icon}
```

系统设定时，对话管理器使用系统文件中的标准"停止"图标。您可通过这个资源标识符号码提供您自己的'ICON'资源来更改这个图标。

使用"停止"警告来告诉用户问题或情况是如此严重以致不能完成动作。"停止"警告一般只有一个按钮 "确定"），因为用户所能做的只是认可该动作不能完成。

亦请参阅

6－9页的图6－5显示了在一个典型“停止”警告中的“停止”图标。除了它在警告栏中包括“停止”图标以外，StopAlert与Alert函数等同。请参阅6－106页对Alert函数的说明，了解关于这两个函数的参数和表现的详细信息。

NoteAlert

若要显示一个左上角有“注意”图标的警告栏（或，如果与警告阶段相适应，发出示警声代替警告栏，或者除警告栏以外还发出示警声），使用NoteAlert函数。

```
FUNCTION NoteAlert(alertID:Integer;
                   filterProc:ModalFilterProcPtr): Integer;
```

alertID 警告资源的资源标识符。如果警告资源丢失，对话管理器就返回到您的应用程序，不创建请求的警告。

filterProc 响应ModalDialog过程不处理的事件的函数的指针。如果您把这个参数设定成NIL，对话管理器就使用标准的事件过滤函数，它允许用户点按Return或Enter键代替点按系统设定按钮。但是，您的应用程序应该提供一个简单的、也允许后台应用程序接收更新事件的事件过滤函数。在这个参数中传递事件过滤函数的指针。

说明

在绘制警告栏内的项目之前，NoteAlert先在左上角：（在局部坐标为［10，20，42，52］的矩形内）绘制“注意”图标，除此之外，NoteAlert函数与Alert函数是相同的。“注意”图标的资源标识符如下：

```
CONST noteIcon=1;  {note icon}
```

系统设定时，对话管理器使用系统文件中的标准“注意”图标。您可通过用这个资源标识符号码提供您自己的‘ICON’资源来更改这个图标。

使用“注意”警告告诉用户出现了小错误，如果对它置之不理也不会发生灾难性的后果。通常这种类型的警告只给出信息，用户通过单击“确定”按钮响应，有时“注意”警告也可能提出一个简单的问题并提供响应选择。

亦请参阅

6－8页的图6－3显示了一个典型“注意”警告中的“注意”图标。除了它在警告栏中包括“注意”图标以外，NoteAlert与Alert函数等同。请参阅6－106页对Alert函数的说明，了解关于这两个函数的参数和表现的详细信息。

CautionAlert

若要显示一个左上角有“当心”图标的警告栏（或，如果与警告阶段相适应，发出示警声代替警告栏，或者除警告栏以外还发出示警声），使用 CautionAlert 函数。

```
FUNCTION CautionAlert  (alertID£∫ Integer;
                       filterProc: ModalFilterProcPtr); Integer;
```

alertID　警告资源的资源标识符。如果警告资源丢失，对话管理器就返回到您的应用程序，不创建请求的警告。

filterProc 响应 ModalDialog 过程不处理的事件的函数的指针。如果您把这个参数设定成 NIL，对话管理器就使用标准的事件过滤函数，它允许用户敲击 Return 或 Enter 键代替单击系统设定按钮。但是，您的应用程序应该提供一个简单的、也允许后台应用程序接收更新事件的事件过滤函数。在这个参数中传递事件过滤函数的指针。

说明

在绘制警告栏内的项目之前，CautionAlert 先在左上角（在局部坐标为 [10, 20, 42, 52] 的矩形内）绘制“当心”图标，除此之外，CautionAlert 函数与 Alert 函数是相同的。“当心”图标的资源标识符如下：

```
COST CautionIcon=2;  {caution icon}
```

系统设定时，对话管理器使用系统文件中的标准“当心”图标。您可通过用这个资源标识符号码提供您自己的‘ICON’资源来更改这个图标。

如果允许一个动作继续进行可能会产生不期望的结果，就使用“当心”警告向用户发出警告信号。给用户提供继续动作（通过单击“确定”按钮）或停止它（通过单击“取消”按钮）的选择。

亦请参阅

6 - 9 页的图 6 - 4 显示了一个典型的“当心”警告中的“当心”图标。除了它在警告栏中包括“当心”图标以外，CautionAlert 与 Alert 函数等同。请参阅 6 - 106 页对 Alert 函数的说明，了解关于这两个函数的参数和表现的详细信息。

GetAlertStage

若要确定最新警告的阶段，使用GetAlertStage函数。GetAlertStage函数与GetAlrtStage函数等同。

```
FUNCTION GetAlertStage:Integer;
```

说明

GetAlertSatge函数返回从0至3的一个数作为最新警告的阶段。例如，如果在某阶段要显示警告栏，您就可使用GetAlertStage函数来保证您的应用程序停用当前活动窗口。

汇编语言信息

全局变量ACount包含这个数。另外，全局变量ANumber包含最新发生的警告的警告资源的资源标识符。

亦请参阅

6－66页的程序清单6－19说明了如何使用GetAlertStage来确定是否根据当前警告阶段停用窗口。6－21页的程序清单6－2说明了如何使用警告资源根据不同的警告阶段指定不同的警告响应。

ResetAlertStage

若要将当前警告阶段复位至第一警告阶段，使用ResetAlertStage过程。ResetAlrtStage过程与ResetAlertStage过程等同。

```
PROCEDURE ResetAlertStage;
```

说明

ResetAlertStage过程将每个警告复位至第一阶段警告。

亦请参阅

6－21页的程序清单6－2说明了如何使用警告资源根据不同警告阶段指定不同的警告响应。

创建和处理对话框

若要创建对话框，您一般使用GetNewDialog函数，它从资源文件的对话资源中获取关于对话的信息。像窗口资源一样，对话资源也将说明信息与您的应用程序代码分隔开，以便修改或转换成其它语音。但是您也可以使用NewDialog和NewColo-rDialog函数，在参数中给它们传递说明信息，来创建对话框。

NewColorDialog返回彩色图形端口的指针，除此之外，NewColorDialog函数与NewDialog函数完全相同。

当您不再需要对话框时，如果您已为对话框的对话记录分配存储器，就使用CloseDialog过程；如果没有就使用DisposeDialog过程。(若仅仅使用户看不见对话框，您可使用窗口管理器过程HideWindow)。

GetNewDialog

若要依据对话资源中的说明创建对话框，使用GetNewDialog函数。

```
FUNCTION GetNewDialog (dialogID: Integer; dStorage: Ptr;
                                     behind: WindowPtr); DialogPtr ;
```

dialogID　　对话资源的资源标识符。如果对话资源丢失，对话管理器就返回到您的应用程序，不创建对话框。

dStorage　　对话记录的存储器的指针。对于定模式对话框和可移动定模式对话框，如果您把这个参数设定成NIL，对话管理器就自动地在您的应用程序堆中给它们分配存储器。然而，对于不定模式对话框，您应该像对待窗口一样给它分配您自己的存储器，否则，您的堆中就可能出现存储碎片。

behind　　一个窗口的指针，对话框将被放置于桌面环境中这个窗口的后面。始终把这个参数设定为窗口指针Pointer (-1)，把对话框送到所有其它窗口的前面。

GetNewDialog　　函数依据对话资源中的信息创建对话记录并返回这个记录的指针。您可在窗口管理器或QuickDraw例程中使用这个指针操纵对话框。如果对话资源规定对话框应该可视，就显示对话框。如果对话资源规定对话框最初应该不可视，就使用窗口管理过程ShowWindow来显示对话框。

如果您提供一个与对话资源具有相同的资源标识符的对话项目色彩表资源，GetNewDialog就使用NewColorDialog函数并返回彩色图形端口的指针。如果没有对话项目色彩表资源，尽管系统软件使用系统系统设定绘制窗口框架，GetNewDialog 使用NewDialog返回一个黑白图形端口的指针。

GetNewDialog的参数dStorage和behind与它们在窗口管理器函数GetNewWindow中的含义相同。始终把behind参数设定成Pointer (-1) 把对话框送到最前面。

对话资源包含对话框的项目列表资源的资源标识符。在调用资源管理器把项目列表资源读进存储器（如果它还不在存储器中）之后，GetNewDialog复制项目列表资源的一个备份并使用这个备份；这样您可能有几个项目相同的对话框。

如果您提供对话项目色彩表资源，GetNewDialog就在把它传递给窗口管理器例程SetWindColor之前复制它，除非对话项目色彩表资源的输入号码单元设定成 - 1（这种情况下则使用系统设定窗口色彩）。GetNewDialog函数通过制作备份使得可在不影响对话框的前提下清除对话项目色彩表资源。

特殊考虑

GetNewDialog函数并不释放资源占用的存储器。因而，您的应用程序应该对对话框使用的所有资源置可清除标记。

如果对话资源或项目列表资源不能被读，函数结果就为NIL。您的应用程序应该进行测试以保证返回NIL后不对警告栏或其项目进行进一步操作。

对于定模式对话框，对话管理器函数ModalDialog捕获所有的事件。这样就阻止您的事件循环接收您的窗口的激活事件。因而，当您使用GetNewDialog创建定模式对话框时如果您的应用程序的窗口是当前活动窗口，您就必须在显示定模式对话框之前显式地停用那个窗口。

如果您的应用程序在后台运行或不可视时您需要显示一个对话框，您就应该使用通知管理器给用户投寄一个通知。通知管理器自动地显示一个包含您所指定的消息的警告栏。您不必使用对话管理器自己创建警告栏。

注意，当您的应用程序在后台时没有任何把信息从用户传返回应用程序的措施。通知管理器提供一个从您的应用程序到用户的单向通讯通道。（尽管您的应用程序有可能确定通知是否被接收）。如果您需要征求用户的信息，使用通知管理器通知用户把您的应用程序切换到前台，在这里用户可以响应您的应用程序展示的对话框。

GetNewDialog 函数使用 NewDialog 或 NewColorDialog，它们均生成全部窗口内容的更新事件。除控制部件外，其它项目都不是马上绘制的。对话管理器调用控制管理器来绘制控制部件，控制管理器就立即绘制这些控制部件。这样控制部件就不会分两次绘制，对话管理器调用窗口管理器过程 ValidRect 用来绘制每个控制部件的环绕矩形。如果您发现绘制控制部件和绘制其它项目之间存在很大的延迟，最好一开始就使对话框不可视，然后再调用窗口管理器过程 SHowWindow 来显示它。

亦请参阅

请参阅 6－23 页的“创建对话框”和 6－61 页的“显示警告栏和对话框”部分了解关于如何使用 GetNewDialog 的讨论和实例。

在本书的“窗口管理器”一章中介绍了 GetNewWindow 和 ShowWindow 过程。在《Inside Macintosh 进程》*(Inside Macintosh : Processes)* 的“通知管理器”一章中介绍了通知管理器。

6－68 页的“针对定模式对话框调整菜单”和 6－73 页的“针对可移动定模式和不定模式对话框调整菜单”部分讨论了当您的应用程序显示对话框时的菜单调整。请参阅 6－37 页的“按钮、复选框、和选项按钮的标题”部分和 6－40 页的“静态文本项目和可编辑文字项目的文字字符串”部分了解关于警告栏中的消息和控制标题的建议。

NewColorDialog

若要创建对话框，您可使用 NewColorDialog 函数，它返回彩色图形端口的指针。通常，您应使用 GetNewDialog 来创建对话框，因为 GetNewDialog 从资源文件的对话资源中获取关于对话框的信息。(与窗口资源相似，对话资源把说明信息与您的应用程序代码隔离开，以便修改或转换成其它语音。) NewCDialog 函数与 NewColorDialog 函数等同。

```
FUNCTION NewColorDialog (dStorage: Ptr; boundsRect : Rect ;
             title : Str255 ; visible : Boolean ;
             procID : Integer ; behind : WindowPtr ;
             goAwayFlag : Boolean ; refCon : LongInt ;
             items : Handle) ; CDialogPtr ;
```

dStorage 对话记录的存储器的指针。对于定模式对话框和可移动定模式对话框，如果您把这个参数设定成NIL，对话管理器就在您的应中程序堆上给它们分配存储器。但对于不定模式对话框，您应该像对待窗口一样分配您自己的存储器，否则您的堆中就可能出现存储碎片。

boundsRect 用全局坐标给出的矩形，它确定对话框的大小和位置；这些坐标指定了对话框的左上角和右下角。

title 用作不定模式或可移动定模式对话框的标题的文字字符串。您可为不包含任何文字的标题栏指定一个空字符串（不是NIL）。

visible 规定是否立即在屏幕上绘制对话框的标志。如果您把这个参数设定成FALSE，就要直到您的应用程序使用窗口管理器过程ShowWindow显示对话框才绘制它。

procID 对话框类型的窗口定义标识符。使用dBoxProc常数来指定定模式对话框，noGrowDocProc常数来指定不定模式对话框，movableDBoxProc常数来指定可移动定模式对话框。

behind 一个窗口的指针，对话框将被放置于桌面环境中这个窗口的后面。如果把这个参数设定为窗口指针Pointer(-1)，就把对话框送到所有其它窗口的前面。

goAwayFlag 规定当一个不定模式对话框是最前面的窗口时在其标题栏中是否应该有关闭格的标志。如果您把这个参数设定成TRUE，当对话管理窗口标题栏中就有关闭格。只有不定模式对话框才应该有关闭格。

refCon 对话管理器用来设定对话框的窗口记录的refCon域的值。您的应用程序可在这里随意存储任何值。例如，您的应用程序可在此存储一个代表对话框类型的号码，也可存储保留对话框的状态信息的记录的指针。您可在任何时候使用窗口管理器过程SetWRefCon来更改对话框的对话记录中的这个值，并且您可使用GetWRefCon函数来确定它的当前值。

items 对话框的项目列表资源的间接指针。您可通过调用资源管理器函数GetResource把项目列表资源读入存储器来取得这个间接指针。当您使用这个间接指针时，使用存储器管理器过程HNoPurge将它置为不可清除。或使用操作系统实用程序HandToHand来复制这个间接指针的拷贝并使用这个备份。

说明

NewColorDialog函数根据它的参数的规定创建对话框并返回新对话框的彩色图形端口的指针。前8个参数（dStorage至refCon）传递给窗口管理器函数NewCWindow，它创建对话框。您可在窗口管理器或QuickDraw例程中使用这个指针来操纵对话框。

对话管理器给对话框使用系统设定窗口色彩。通过使用系统设定色彩，您可保证您的应用程序的界面与Finder和其它应用程序的一致。但是，如果您实在觉得有必要打破这种一致性，您就可使用窗口管理器过程SetWinColor来使用您自己的对话项目色彩表资源，这个资源指定了系统设定色彩以外的其它彩色。不过，要意识到在您的警告和对话框中的非标准色彩最初可能会迷惑您的用户。

窗口管理器为彩色对话框创建一个辅助窗口记录。您可通过窗口管理器函数GetAuxWin存取这个记录。(辅助窗口记录的dialogCItemhandle域指向对话框的项目色彩表资源。)如果对话框的内容色彩不是白色，建议把visible标志设定成FALSE然后调用NewColorDialog。在它装入彩色表和彩色项目列表资源之后，如果对话框是最前面窗口就使用窗口管理器过程ShowWindow来显示对话框。如果对话框是处在后台的不定模式对话框，就使用窗口管理器过程ShowHide来显示它。

当在boundsRect参数中指定对话框的大小和位置时，您一般应该尽量使对话框处于用户正在工作的屏幕或窗口的左右边沿之间的中心位置，或任何更合适的位置。当您在主屏幕上放置对话框时还要保证对话框的顶部(包括不定模式和可移动定模式对话框的标题栏)位于菜单栏的下面。您可使用菜单管理器函数GetMBarHeight来确定菜单栏的高度。

特殊考虑。

对于定模式对话框，对话管理器函数ModalDialog俘获所有事件。这样就防止您的事件循环接收您的窗口的激活事件。因而，当您使用NewColorDialog创建定模式对话框时如果您的应用程序的窗口是当前活动窗口，您就必须在显示定模式对话框之前显式地停用那个窗口。

如果您的应用程序在后台运行时或不可视时您需要显示一个对话框，您就应该使用通知管理器给用户投寄一个通知。通知管理器自动地显示一个包含您所指定的消息的警告栏。您不必使用对话管理器自己创建警告栏。

注意，当您的应用程序在后台时没有任何把信息从用户传递回应用程序的措施(尽管您的应用程序有可能确定通知是否被接收)。通知管理器提供一个从您的应用程序到用户的单向通讯通道。如果您需要征求用户的信息，使用通知管理器通知用户把您的应用程序送到前台，在这里用户可以响应您的应用程序展示的对话框。

NewColorDialog函数生成全部窗口内容的更新事件。因而除控制部件外，其它项目都不是马上绘制的。对话管理器调用控制管理器来绘制控制部件，控制管理器就立即绘制这些控制部件。使得控制部件不分两次绘制，对话管理器调用窗口管理器过程ValidRect用来绘制每个控制部件的环绕矩形。如果您发现绘制控制部件和绘制其它项目之间存在很大的延迟，最好一开始就使对话框不可视，然后再调用窗口管理器过程ShowWindow来显示它。

亦请参阅

本书的“窗口管理器”一章对窗口管理器例程作了介绍。《Inside Macintosh进程》*(Inside MacintoshProcesse)*的“通知管理器”一章介绍了通知管理器。请参阅《Inside Macintosh内存》*(Inside Macintosh : Mernory)*了解关于HNoPurge的说明。请参阅《Inside Macintosh操作系统实用程序》*(Inside Macintosh : Operating System Utilities)*了解关于HandToHand的说明。

6－68页的“针对定模式对话框调整菜单”部分和6－73页的“针对可移动定模式和不定模式对话框调整菜单”部分讨论了当您的应用程序显示对话框时的菜单栏调整。请参阅6－37页的“按钮、复选框和选项按钮的标题”部分和6－40页的“静态文字和可编辑文字项目的文字字符串”部分，了解关于对话框中的消息和控制标题的建议。《Inside Macintosh工具箱扩展》*(Inside Macintosh : More Macintosh Toolboox)*的“资源管理器”一章介绍了GetResource函数。

NewDialog

若要创建对话框，您可使用NewDialog函数，它返回黑白图形端口的指针（尽管系统软件使用系统设定窗口彩色绘制对话框的窗口框架）。通常，您应该使用GetNewDialog来创建对话框。GetNewDialog从资源文件的对话资源中获取关于对话框的信息。（与窗口资源相似，对话资源把说明信息与您的应用程序代码隔离开，以便修改或转换成其它语言。）

NewDialog返回黑白图形端口的指针，除此之外，NewDialog函数与NewColorDialog函数等同。请参阅6－115页对NewColorDialog的讨论，了解那些您也传递给NewDialog的参数的说明。

```
FUNCTION NewDialog (dStorage : Prt ; boundsRect : Rect ;
                    title : Str255 ; visible : Boolean ;
                    procID : Integer ; behind : WindowPtr ;
                    goAwayFlag : Boolean ; refCon : LongInt ;
                    items : Handle) : DialogPtr ;
```

说明

NewDialog函数根据它的参数的规定创建对话框并返回新对话框的黑白图形端口的指针。前8个参数（dStorage至refCon）传递给窗口管理器函数NewWindow，由它创建对话框。

当在boundsRect参数中指定对话框的大小和位置时，您一般应该尽量使对话框处于用户正在工作的屏幕或窗口的左右边沿之间的中心位置，或任何更合适的位置。当您在主屏幕上放置对话框时还要保证对话框的顶部(包括不定模式和可移动定模式对话框的标题栏)位于菜单栏的下面。您可使用菜单管理器函数GetMBarHeight来确定菜单栏的高度。

亦请参阅

如果您使用对话项目色彩表资源更改系统设定窗口色彩，就使用NewColorDialog函数，它返回彩色图形端口的指针。请参阅6－115页关于NewColorDialog的说明，了解NewDialog函数和NewColorDialog函数共同的辅助信息。

CloseDialog

如果您给一个对话框的对话记录分配了存储器，若要消除这个对话框，就使用CloseDialog过程。

```
PROCEDURE CloseDialog (theDialog : DialogPtr);
```

theDialog　　对话记录的指针。

说明

CloseDialog在屏幕上取消一个对话框并在窗口列表中删除它。CloseDialog释放下述对象占用的存储器：

- 对话框相关联的数据结构（如对话框的结构、内容和更新区域）
- 对话框中的所有项目（图片和图标除外，因为它们可能被其它资源共享）和与它们相关的所有数据结构

一般说来，当您创建不定模式对话框时您应该给它的对话记录提供存储器。(您可让对话管理器给定模式和可移动定模式对话框提供存储器。)然后当用户点按关闭格或在文件菜单中选择“关闭”时，您应该使用CloseDialog来关闭不定模式对话框。

由于CloseDialog并不处理对话资源或项目列表资源，所以使这些资源可清除是很重要的。不像GetNewDialog、NewColorDialog并不使用项目列表资源的备份。因而，如果您使用NewColorDialog创建对话框，您就要使用CloseDialog使项目列表资源保持在存储器中（即使您并不给出存储器的指针）。

亦请参阅

如果您让对话管理器给对话框分配存储器（通过在 GetNewDialog，NewColorDialog 或NewDialog函数的 dStorage 参数中传递NIL ），就使用DisposeDialog 过程而不用CloseDialog过程。下面将介绍DisposeDialog过程。

DisposeDialog

若要消除一个对话管理器为它提供存储器的对话框，就使用DisposeDialog过程。DisposDialog过程与DisposeDialog过程等同。

```
PROCEDURE DisposeDialog ((theDialog : DialogPtr) ;
```

theDialog　　对话记录的指针。

说明

DisposeDialog过程调用CloseDialog过程，还释放对话框的项目列表资源和对话记录占用的存储器。如果您在GetNewDialog，NewColorDialog和NewDialog的参数dStorage传递NIL，当您结束对话框时就调用DisposeDialog。

一般情况下，您的应用程序不应该给定模式对话框或可移动定模式对话框的对话记录分配存储器。在这些情形中当用户单击“确定”或“取消”按钮时您的应用程序应该使用DisposeDialog。

亦请参阅

如果您给对话框分配存储器（例如，通过在GetNewDialog，NewColorDialog或NewDialog的dStorage参数中传递一个指针），就使用CloseDialog代替DisposeDialog。6－119页介绍了CloseDialog。

操纵警告栏和对话框中的项目

大多数情况下，您在资源文件中定义警告栏和对话框之后就不再对它作任何更改。如果您需更改它们，就使用这部分介绍的对话管理器例程。

对于大部分项目操纵，首先调用GetDialogItem过程来获取关于项目的信息。然后您可使用其它例程来操纵这个项目，如果您使用任何其它例程来更改项目就使用SetDialogItem过程。您也必须使用SetDialogItem来安装您自己的应用程序定义的任何绘图例程。如果您使用SetDialogItem，最好使对话框最初不可视，然后相应地更改项目，最后通过使用窗口管理器过程ShowWindow使对话框可视。（若想了解关于操纵警告或对话框中的文字的信息，请参阅6－129页的“处理警告栏和对话框中的文字”部分。）

通过使用AppendDITL和ShortenDITL过程，您可动态地给对话框添加项目和取消其项目。如果您的多个对话框共享一个项目列表资源，这些过程就显得特别有用，因为您可以随后使用AppendDITL和ShortenDITL相应地给某个对话框添加项目或取消其中的项目。您一般先使这样的对话框不可视，相应地使用AppendDITL和ShortenDITL过程，然后使用窗口管理过程ShowWindow使对话框变成可视。

GetDialogItem

若要获取一个项目的间接指针使得您可以操纵这个项目(例如，确定它的当前值、更改它或安装定义应用程序的项目的绘图过程的指针)，使用GetDialogItem过程。GetDItem过程与GetDialogItem过程等同。

```
PROCEDURE GetDialogItem (theDialog : DialogPtr ; itemNo : Integer ;
                         VAR itemType : Integer ; VAR item : Handle ;
                         VAR box : Rect) ;
```

theDialog　　对话记录的指针。

itemNo　　一个号码，对应项目在对话框的项目列表资源中的位置。

itemType　　表示在itemNo参数中请求的项目的类型的值。您可任意使用下列常数来确定这个参数返回的值。

item对于一个应用程序定义的绘图过程，它是绘图过程的指针（强制转换为间接指针），返回给 itemNo 参数中指定的项目；对于所有其它项目类型，它是项目的间接指针。

box　显示矩形（用对话框的局部坐标表示），返回给 itemNo 参数指定的项目。

说明

GetDialogItem过程在它的参数中返回关于指定对话框的项目列表资源中标号为 itemNo 的项目的信息，这些信息如下：项目类型（在 itemType 参数中）；在 item 参数中返回项目的间接指针（或者，对于应用程序定义的绘图过程，返回过程指针）；在 box 参数中返回项目的显示矩形。

对于大部分项目操纵，首先使用 GetDialogItem 过程获取关于项目的信息。您可使用其它例程，如 GetDialogItemText 和 SetDialogItem，来确定和改变这个项目的值。

亦请参阅

6 — 49 页的程序清单 6 — 12 说明了如何结合使用 GetDialogItemText 和 GetDialogItem 来检索用户在可编辑文字项目中输入的文字。6 — 58 页的程序清单 6 — 16 说明了如何结合使用 GetDialogItem 和 SetDialogItem 把应用程序定的项目的绘图过程安装到对话框内。6 — 83 页的程序清单 6 — 26 说明了如何使用 GetDialogItem 来确定对话框中的复选框的当前值。

SetDialogItem

使用 GetDialogItem 过程获取一个对话框项目的间接指针之后，使用 SetDialogItem 过程来设定或更改项目。SetDItem 过程与 SetDialogItem 过程等效。

```
PROCEDURE SetDialogItem (theDialog : DialogPtr ; itemNo : Integer ;
                         itemType : Integer ; item : Handle ;
                         box : Rect) ;
```

theDialog 对话记录的指针。

itemNo　一个号码，对应项目在对话框的项目列表资源中的位置。

itemType　表示在 itemNo 参数中的项目的类型的值。若要指定这个参数值，您可使用 6 — 121 页中为 GetDialog 过程的 itemType 参数列出的任何常数。

item　　对于定义应用程序的项目，它是itemNo参数指定的项目的绘图过程的指针（强制转换成间接指针）；对于其它项目类型，是项目的间接指针。

box　　为itemNo参数指定的项目的显示矩形（用对话框的局部坐标表示）。

说明

SetDialogItem设定由itemNo参数指定的在指定对话框中的项目。这个过程安装项目但不绘制项目；您一般使用SetDialogItem创建一个不可视对话框，然后使用窗口管理器过程ShowWindow来绘制对话框及其项目。

亦请参阅

6－58页的程序清单6－16说明了如何使用SetDialogItem来安装一个应用程序定义的绘图过程。本书的“窗口管理器”一章介绍了ShowWindow过程。

HideDialogItem

您可以使用HideDialogItem过程使对话框中的项目不可视，但您极少需要这么做。HideDItem过程与HideDialogItem过程等同。

```
PROCEDURE HideDialogItem (theDialog : DialogPtr ; itemNo : Integer) ;
```

theDialog　　对话记录的指针。

itemNo　　一个号码，对应一个项目在对话框的项目列表资源中的位置。

说明

HideDialogItem通过给予项目一个屏幕之外的显示矩形来隐藏项目。具体地说，如果项目的显示矩形的左坐标小于8192（16进制＄2000），HideDialogItem就给矩形的左、右坐标均添加16，384（16进制＄4000）。如果项目已经是隐藏的（即左坐标大于8192），HideDialogItem就什么也不做。若要重新显示被HideDialogItem隐藏的项目，您可使用ShowDialogItem过程。

特殊考虑

如果您的应用程序需要显示许多相似的对话框（其中有一两个例外），使用 AppendDITL 和 ShortenDITL 过程来修改共同的单元会比使用 HideDialogItem 和 ShowDialogItem 过程简便些。

静态文本项目的矩形必须至少有一个字符那么宽。

您一般不应使用 HideDialogItem 使可编辑文字项目不可视，因为当用户敲 Tab 键时，对话管理器试图将光标移至隐藏的可编辑文字项目，在那里放置用户随后所敲的键码。

ShowDialogItem

若要重新显示被 HideDialogItem 过程隐藏的项目，请调用 ShowDialogItem 过程。ShowDItem 过程与 ShowDialogItem 过程等同。

```
PROCEDURE ShowDialogItem (theDialog : DialogPtr ; itemNo : Integer) ;
```

theDialog 对话记录的指针。

itemNo　　项目号码，对应于项目在对话框的项目列表资源中的位置。

说明

ShowDialogItem 过程通过还原调用 HideDialogItem 之前项目的显示矩形来重新显示项目。具体地说，如果项目的显示矩形的左坐标大于8192，ShowDialogItem 就从矩形的左坐标和右坐标中都减去16,384。如果项目已经可视(即左坐标小于8192)，ShowDialogItem 就什么也不做。

ShowDialogItem 过程将包含项目的矩形添加至更新区域以便绘制它。注意，如果项目是您在控制(`CNTL')资源中定义的一个控制部件，添加至更新区域的矩形就是在控制资源中定义的矩形，而不是在项目列表资源中定义的显示矩形。如果项目是可编辑文字项目，ShowDialogItem 就通过调用 TextEdit 过程 TEActivate 来激活它。

FindDialogItem

若要确定在对话框内某个特殊位置上的项目的项目号码，使用FindDialogItem函数。FindDItem函数与FindDialogItem函数等同。

```
FUNCTION FindDialogItem (theDialog : DialogPtr ; thePt : Point) : Integer ;
```

theDialog　　对话记录的指针。

thePt　　用对话框局部坐标指定的一个点。

说明

如果参数thePt指定的点位于一个项目内，FindDialogItem就返回此项目号码，这个号码对应于对话框的项目列表资源中那个项目的位置。如果这个点不位于项目的矩形内，FindDialogItem就返回－1。如果有重叠的项目，FindDialogItem就返回项目列表资源中包含该点的第一个项目的项目号码。

当光标在一特别项目上要改变光标时，这个函数就有了用武之地。

对于项目列表资源中的第一个项目FindDialogItem函数返回0，第二个项目返回1，如此类推。为了在调用GetDialogItem或SetDialogItem过程之前获取正确的项目号码，给FindDialogItem的函数结果加1，如下所示：

```
theItem := FindDialogItem (theDialog : thePoint) + 1 ;
```

注意，FindDialogItem既返回有效项目的项目号码，也返回无效项目的项目号码。

AppendDITL

当您的应用程序正在运行时，若要给一个现存的对话框添加项目，使用AppendDITL过程。

```
PROCEDURE AppendDITL (theDialog : DialogPtr ; theDITL : Handle ;
                      theMethod : DITLMethod) ;
```

theDialog 对话记录的指针。您将把参数theDITL指定的项目列表资源添加至这个对话记录。

theDITL　项目列表资源的间接指针，您要把它的项目增补到对话框中。

theMethod在现存的对话框中新项目显示的方式。您可传递一个负值使增补的项目从现存对话框内特别的项目处开始偏移。您也可传递下列任何常数：

说明

AppendDITL过程将参数theDITL指定的项目列表资源中的项目添加至对话框的项目。如果几个对框共享一个项目列表资源，这个过程就显得特别有用，因为您可以使用AppendDITL来添加适合于单个对话框的项目。如果您要将一个项目列表资源的项目添加到对话框中，您的应用程序可使用资源管理器函数GetResource来获取这个项目列表资源的间接指针。

在参数 theMethod 中，您可以指定增补新项目的方法，具体情况如下：

- 如果您使用overlayDITL常数，AppendDITL就把增补的项目叠加到对话框上。这就是说，AppendDITL把增补项目的显示矩形的坐标（在它们的项目列表资源中指定的）解释 为对话框内的局部坐标
- 如果您使用appendDITLRight常数，AppendDITL就把项目增补到对话框的右边，相对于对话框的右上坐标定位增补项目的显示矩形。AppendDITL过程能够自动扩展对话框以便容纳新的对话项目
- 如果您使用appendDITLBottom常数，AppendDITL就把项目增补到对话框的底部，相对于对话框的左下坐标定位增补项目的显示矩形。AppendDITL过程自动地扩展对话框以容纳新的对话项目
- 您也可以在参数theMethod中传递一个负值，相对一个现存的项目来增补项目列表，这个负值的绝对值解释为对话框中的一个项目，新项目就相对这个项目来定位。例如，如果您传递－2，增补项目的显示矩形就相对在对话框中项目号码为2的项目的左上角偏移

一般调用AppendDITL过程使对话框不可视，然后通过使用窗口管理器过程ShowWindow使对话框可视。

特殊考虑

AppendDITL过程修改对话框的内容（例如放大它）。如果希望使用对话框未修改的版本，您的应用程序应该使用资源管理器过程ReleaseResource来释放增补的项目列表资源所占用的存储器。否则，如果您的应用程序再次调用AppendDITL给那个对话框添加项目，对话框将仍然保持上次调用AppendDITL所作的修改一例如，若您原来使用的是appendDITLBottom常数，它在底部仍然较长。

AppendDITL 过程可以在 System 7 和通信工具箱中以前版本的使用。您在调用 AppendDITL 之前，应该通过调用带有 gestaltDITLExtPresen 选定项的 Gestalt 函数确保它是可用的。您需要测试 response 参数中的 gestaltDITLExtPresent 常数标志位。如果该数位已被设定，AppendDITL 就可以使用。

亦请参阅

6-54 页的列表 6-13 和 6-55 页的列表 6-14 说明了 AppendDITL 典型的使用情况。6-52 页的图 6-29 显示了使用 overlayDITL 常数的结果。6-52 页的图 6-31 显示了使用 appendDITLBottom 常数的结果。6-53 页的图 6-32 显示了在参数 theMethod 中使用负数的结果。

《Inside Macintosh 工具箱扩展》*(Inside Macintosh : More Macintosh Toolbox)* 一书中的"资源管理器"一章说明了 GetResource 和 ReleaseResource 例程。Gestalt 函数在《Inside Macintosh 操作系统实用程序》*(Inside Macintosh : Operating System Utilities)* 一书中的"格式塔管理器"一章中作了说明。有关 ShowWindow 的信息，请参阅本书的"窗口管理器"一章。

ShortenDITL

为了在您的应用程序运行时，从现有对话框中删除项目，应使用 ShortenDITL 过程。

```
PROCEDURE ShortenDITL  (theDialog : DialogPtr ;
                        numberItems : Integer) ;
```

theDialog　　对话记录的一个指针。

numberItems　　要删除的项目的数目（从项目列表资源的最后一个项目开始）。

说明

ShortenDITL 过程从对话框中删除指定数目的项目。如果几个对话框共享一个项目列表，这个过程就特别有用。因为您可以在必要时，使用 ShortenDITL 删除单个对话框中的项目。

通常，您创建的是一个不可见的对话框，要使对话框可见，应调用 ShortenDITL 过程，然后使用窗口管理器的过程 ShowWindow。注意，ShortenDITL 不会自动地调整对话框的大小；如果需要调整对话框的大小，您可以使用窗口管理器的过程 SizeWindow。

特殊考虑

ShortenDITL 过程可以在 System 7 和以前的版本的通信工具箱中使用。您在调用 ShortenDITL 之前，应该通过使用带有 gestaltDITLExtPresen 选定项的 Gestalt 函数确保它是可用的。您需要用 gestaltDITLExtPresent 常数测试 response 参数中的标志位。如果该标志位是设定的，ShortenDITL 就可以使用。

亦请参阅

您可以使用下面说明的 CountDITL 函数确定对话框的项目列表资源中的项目数。有关 ShowWindow 和 SizeWindow 过程的信息，请参阅本书的“窗口管理器”一章。Gestalt 函数在《Inside Macintosh 操作系统实用程序》*(Inside Macintosh : Operating System Utilities)* 一书的“格式塔管理器”一章中说明。

CountDITL

您可以使用 CountDITL 函数确定对话框中的项目数。

```
FUNCTION CountDITL  (theDialog : DialogPtr) : Integer ;
```

theDialog　　对话记录的一个指针。

说明

CountDITL 返回一个对话框中的当前项目数。通常您使用 ShortenDITL 和 CountDITL 函数，从对话框中删除项目。

特殊考虑

CountDITL 过程可以在 System 7 和以前的版本的通信工具箱中使用。您在调用 CountDITL 之前，应该通过使用带有 gestaltDITLExtPresen 选定项的 Gestalt 函数确保它是可用的。您需要用 gestaltDITLExtPresent 常数测试 response 参数中的标志位。如果该标志位是设定的，CountDITL 就可以使用。

亦请参阅

Gestalt 函数在《Inside Macintosh 操作系统实用程序》*(Inside Macintosh : Operating System Utilities)* 一书的“格式塔管理器”一章中说明。

处理警告栏和对话框中的文字

对话管理器提供了几个操纵文字的例程。您可以使用 ParamText 过程在警告栏和对话框的静态文本项目中动态地提供文字字符串，例如文稿的标题。对于在静态文字和可编辑文字项目中确定和改变文字，GetDialogItemText 和 SetDialogItemText 过程非常有用。您可以使用 SetDialogItemText 过程在一个可编辑文字项目中选定和醒目显示文字。

当一个包含可编辑文字项目的对话框有效时，应使用 DialogCut 过程处理编辑命令“剪”，使用 DialogCopy 过程处理命令“复制”，使用 DialogPaste 过程处理命令“贴”，以及使用 DialogDelete 过程处理命令“清除”。

每当您确定在不定模式对话框或可移动定模式对话框中发生了一个事件后，就可以使用 DialogSelect 函数在可编辑文字项目中自动处理键按下事件（请参阅6-139 页）。ModalDialog 过程使用 DialogSelect 处理定模式对话框的可编辑文字项目中的键按下事件。

ParamText

为了在您的应用程序运行时，在警告栏和对话框的静态文本项目中替代文字字符串，应使用 ParamText 过程。

```
PROCEDURE ParamText  (param0 : Str255 ; param1 : Str255 ;
                      param2 : Str255 ; param3 : Str255) ;
```

param0　一个文字字符串，它在以后所创建的警告栏和对话框的静态文本项目中替代特殊字符串^0。

param1　一个文字字符串，它在以后所创建的警告栏和对话框的静态文本项目中替代特殊字符串^1。

param2　一个文字字符串，它在以后所创建的警告栏和对话框的静态文本项目中替代特殊字符串^2。

param3　一个文字字符串，它在以后所创建的警告栏和对话框的静态文本项目中替代特殊字符串^3。

说明

ParamText 过程在以后所创建的警告栏和对话框的静态文本项目中用您作为参数传送的字符串替代字符串 ^0 到 ^3。如果不使用参数应传送空字符串（不是 NIL）。

特殊考虑

ParamText 中使用的字符串储存在低端存储器全局变量 DAStrings 中，它指定了对话管理器使用的一组字符串间接指针。

如果用户在您的应用程序的(存储器)分区中启动桌面附件,而且该桌面附件调用 ParamText 过程,它就可能改变应用程序对话框中的文字。

在不定模式对话框使用 ParamText 过程必须非常小心。如果一个使用 ParamText 的不定模式对话框在屏幕上,而您同时又显示另一个也使用 ParamText 的警告栏或对话框,两个方框都将受到最近一次调用 ParamText 的影响。

您传送给 ParamText 的字符串不能包含特殊字符串 ^0 到 ^3,否则在 7.1版之前的系统软件中,该过程将陷入无止境的替代循环中。

注意,为了方便转换成其它语言,您应该在资源文件中储存文字字符串;因此,ParamText 的最佳用途是提供用户指定的文字字符串,例如文稿名称。为了在您本地化应用程序时避免语法和句子结构的问题,您应该使用 ParamText 在每个屏幕消息中只提供一个文字字符串。

亦请参阅

6-47 页的列表 6-9 和 6-48 页的列表 6-10 给出了如何使用 ParamText 为您的告警框和对话框提供用户当前文稿的标题。如果您需要在应用程序运行时,为一个可编辑文字项目提供系统设定文字字符串,应使用 SetDialogItemText。SetDialogItemText 过程还允许您设定或改变静态文本项目的整个文字字符串。

GetDialogItemText

在使用 GetDialogItem 过程获得对话框中的可编辑文字项目或静态文本项目的一个间接指针后,您可以使用 GetDialogItemText 过程获得包含在该项目中的文字字符串。GetDialogItemText 过程和 GetItext 过程是一样的。

```
PROCEDURE GetDialogItemText  (item: Handle; VAR text: Str255);
```

item 对话框中的可编辑文字项目或静态文本项目的一个间接指针。

text 包含在该项目中的文字。

说明

GetDialogItemText 过程在 text 参数中返回指定的可编辑文字项目或静态文本项目的文字。

特殊考虑

如果用户在一个可编辑文字项目中键入的文字超过 255 个字符,GetDialogItemText 只返回前面 255 个字符。

亦请参阅

6-49 页的列表 6-12 说明了如何使用 GetDialogItemText 过程检索用户键入一个可编辑文字项目的文字。

SetDialogItemText

在使用 GetDialogItem 过程获得对话框中的可编辑文字项目或静态文本项目的一个间接指针后，您可以使用 SetDialogItemText 过程显示该项目中的一个特定的文字字符串。SetDialogItemText 过程和 SetItext 过程是一样的。

```
PROCEDURE SetDialogItemText  (item: Handle; text: Str255);
```

item 对话框中的可编辑文字项目或静态文本项目的一个间接指针。

text 要在该项目中显示的文字。

说明

SetDialogItemText 过程在指定的项目中放置指定的文字，并绘制该项目。当您的应用程序运行时，对于为一个可编辑文字项目提供系统设定文字字符串，这个过程是很有用的。

特殊考虑

为了方便转换成其它语言，所有字符串都应该储存在资源文件中。

亦请参阅

对于静态文本项目，如果当您的应用程序运行时，您只需要确定并提供一个文字字符串的一部分，6-129 页中介绍了 ParamText 过程。

SelectDialogItemText

若要选定及醒目显示一个可编辑文字项目中的文字，应使用 SelectDialogItemText 过程。SelectDialogItemText 过程和 SelIText 过程是一样的。

```
PROCEDURE SelectDialogItemText (theDialog: DialogPtr;
                                             itemNo: Integer;
                                             strtSel: Integer;
                                             endSel: Integer);
```

theDialog 对话记录的一个指针

itemNo　一个数字,它相应于一个可编辑文字项目在对话框项目列表资源中的位置。

strtSel　一个数字，它代表所选定项的首字符的位置。

endSel　一个数字，它代表所选定项的末字符的后面一个位置。

说明

如果 itemNo 参数中的项目是包含文字的一个可编辑文字项目，SelectDialogItemText 过程把文字选定范围设定为由 strtSel 参数指定的字符位置开始，直至（但不包括）endSel 参数指定的字符位置。只要 strtSel 不等于 endSel，就醒目显示选定的范围。如果 strtSel 等于 endSel，就显示一个闪烁的垂直条，表示该点是一个插入点。如果可编辑文字项目不包含文字，SelectDialogItemText 显示插入点。

如果您要显示全部文字，应该在 strtSel 参数中指定 0，在 endSel 参数中指定 32767。

例如,如果用户在可编辑文字项目的输入一个不可接受的字符串,您的应用程序可以显示一个警告栏，报告这个问题，然后，使用 SelectDialogItemText 选定域中的全部文字，以便用户在重新输入时这可以被新的输入项直接代替。如果没有这个过程，用户必须在输入新的输入项之前，选定原来的内容。

亦请参阅

有关文字选定范围和字符位置的细节，请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书中的“TextEdit”一章。

DialogCut

当包含一个可编辑文字项目的对话框有效时，应使用DialogCut过程处理编辑命令“剪切”。DialogCut 过程和 DlgCut 过程是一样的。

```
PROCEDURE DialogCut  (theDialog: DialogPtr);
```

theDialog　　对话记录的一个指针。

说明

DialogCut 过程检查对话框是否有可编辑文字项目，如果有，就调用 TextEdit 的过程 TECut 用于选定的文字。当在编辑菜单中处理鼠标按下事件时，您的应用程序应该首先确定对话框是否为最前面的窗口，如果是才调用这个例程。

亦请参阅

有关您的应用程序显示对话框时，允许存取菜单的更详细信息，请参阅 6-68 页开始的“调整定模式对话框的菜单”，以及 6-73 页的“调整可移动定模式对话框和不定模式对话框的菜单”两节。《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书的“TextEdit”一章对 TECut 过程作了说明。

DialogCopy

当包含一个可编辑文字项目的对话框有效时，应使用DialogCopy过程处理编辑命令“复制”。DialogCopy 过程和 DlgCopy 过程是一样的。

```
PROCEDURE DialogCopy  (theDialog :  DialogPtr) ;
```

theDialog　对话记录的一个指针。

说明

DialogCopy 过程检查对话框是否有可编辑文字项目，如果有，就调用 TextEdit 的过程 TECopy 用于选定的文字。当在编辑菜单中处理鼠标按下事件时，您的应用程序应该首先确定对话框是否为最前面的窗口，如果是才调用这个例程。

亦请参阅

有关您的应用程序显示对话框时，允许存取菜单的更详细信息，请参阅 6-68 页开始的“调整定模式对话框的菜单”，以及 6-73 页的“调整可移动定模式对话框和不定模式对话框的菜单”两节。《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书的“TextEdit”一章对 TECopy 过程进行了说明。

DialogPaste

当包含一个可编辑文字项目的对话框有效时，应使用 DialogPaste 过程处理编辑命令“贴”。DialogPaste 过程和 DlgPaste 过程是一样的。

```
PROCEDURE DialogPaste  (theDialog :  DialogPtr) ;
```

theDialog　对话记录的一个指针。

说明

DialogPaste过程检查对话框是否有可编辑文字项目，如果有，就把TextEdit的过程TEPaste用于选定的文字。当在编辑菜单中处理鼠标按下事件时，您的应用程序应该首先确定对话框是否为当前活动窗口，在适用时才调用这个例程。

亦请参阅

有关您的应用程序显示对话框时，允许存取菜单的更详细信息，请参阅 6-68 页的“调整定模式对话框的菜单”，以及6-73页的“调整可移动定模式对话框和不定模式对话框的菜单”两节。《Inside Macintosh文本》*(Inside Macintosh : Text)* 一书的“TextEdit”一章对TEPaste 过程进行了说明。

DialogDelete

当包含一个可编辑文字项目的对话框有效时，应使用DialogPaste过程处理编辑命令“清除”。DialogDelete过程和DlgDelete过程是一样的。

```
PROCEDURE DialogDelete  (theDialog : DialogPtr) ;
```

theDialog 对话记录的一个指针。

说明

DialogDelete过程检查对话框是否有可编辑文字项目，如果有，就把TextEdit 的过程TEDelete用于选定的文字。当在编辑菜单中处理鼠标按下事件时，您的应用程序应该首先确定对话框是否有效，满足条件才调用这个例程。

亦请参阅

有关您的应用程序显示对话框时，允许存取菜单的更详细信息，请参阅 6-68 页的“调整定模式对话框的菜单”，以及6-73页的“调整可移动定模式对话框和不定模式对话框的菜单”两节。《Inside Macintosh文本》*(Inside Macintosh : Text)* 一书的“TextEdit”一章对TEDelete 过程进行了说明。

处理对话框中的事件

处理警告栏中的事件是非常简单的：在您调用一个警告栏后，对话管理器就自动地调用ModalDialog过程，为您处理大多数事件。为了处理定模式对话框中的事件，您的应用程序必须在显示对话框后，明确调用ModalDialog过程。在这两种情况下，只要单击一个启用的项目，对话管理器都返回该对话项目编号。然后您将对单击项目作出适当的响应。对于警告栏和定模式对话框，您都应该提供一个简单的事件过滤函数，它允许其它窗口响应更新事件，并允许您的警告栏或对话框响应几个键的键按下事件，例如Return键、Enter键和Esc键。

您可以使用正常的事件处理代码确定事件是否发生在一个可移动定模式对话框或不定模式对话框中，您也可以使用IsDialogEvent函数了解它们是否需要作为对话框的一部分来处理。只要您确定了事件是发生在一个不定模式对话框或可移动定模式对话框中的，就可以使用DialogSelect函数自动地处理可编辑文字项目中的键按下事件，自动处理更新事件和激活事件，并且报告用户点按的对话项目。然后您对用户单击有效项目进行适当的响应。您也可以不求助于对话管理器，而使用控制管理器、TextEdit和窗口管理器的例程（例如 FindWindow，BeginUpdate，FindControl，TrackControl和 TEClick）处理这些事件。

ModalDialog

若要在您显示一个定模式对话框时处理事件，应使用ModalDialog过程。

```
PROCEDURE ModalDialog  (filterProc : ModalFilterProcPtr ;
                        VAR itemHit : Integer) ;
```

filterProc 事件过滤函数的一个指针。

itemHit 项目编号，它代表选定的项目在有效的定模式对话框的项目列表资源中的位置。

说明

在显示了一个定模式对话框后，就应该立即调用ModalDialog过程。ModalDialog过程假定定模式对话框是作为当前端口显示的，ModalDialog过程不断地处理该端口中的事件，直至发生涉及对话框的有效项目的一个事件，例如单击一个选项按钮。如果该事件是一个在对话框内容区域以外的鼠标按下事件，ModalDialog就发出系统示警声，并获得下一个事件。在收到涉及有效项目的一个事件后，ModalDialog在itemHit参数中返回它的项目编号。然后，您的应用程序应该对该项目中的事件进行适当的响应。您的应用程序应该继续调用ModalDialog，直至用户选定了“确定”或“取消”按钮。

ModalDialog在处理对话框内的事件前，把它传送给filterProc参数中所指向的事件过滤函数。当事件过滤函数返回FALSE时，ModalDialog就处理这个事件。如果事件过滤函数处理这个事件，它就返回TRUE，ModalDialog 将不再对该事件进行处理。

如果您把filterProc参数设定为NIL，就执行标准事件过滤函数。当用户按下Return键或Enter键时，标准事件过滤函数返回TRUE，并使得ModalDialog返回项目编号 1，这是系统设定按钮的编号。但是，您的应用程序应该提供一个简单的事件过滤函数，它具有如下功能:

- 如果用户按下Return键或Enter键，就返回TRUE和系统设定按钮的项目编号
- 如果用户按下Esc键和“命令” + “.”组合键，就返回TRUE和取消按钮的项目编号
- 响应更新事件(这使得后台应用程序能够接收更新事件)更新您的窗口并返回FALSE
- 对于您的事件过滤函数不处理的所有事件，均返回FALSE

您可以在大多数或全部警告栏和定模式对话框中使用同样的事件过滤函数。

您也可以使用filterProc参数中指定的事件过滤函数测试及响应等效命令组合键和更复杂的事件，例如用户在应用程序定义的一个项目中拖拽光标的事件。

为了处理事件，ModalDialog调用IsDialogEvent函数。如果IsDialogEvent的结果是TRUE，ModalDialog就调用DialogSelect函数处理该事件。只要事件过滤函数不返回TRUE，ModalDialog按下述方式处理事件:

- 对于对话框的激活事件或更新事件，ModalDialog激活或者更新它的窗口

- 如果用户按下鼠标按钮的同时，光标在一个可编辑文字项目中，ModalDialog就适当地响应鼠标活动，即显示一个插入点或是选定文字。如果发生了一个键按下事件，并且发生在一个可编辑文字项目，ModalDialog就使用TextEdit自动处理文字输入和编辑。如果可编辑文字项目有效，ModalDialog在收到鼠标按下事件或键按下事件后，返回它的项目编号。在正常情况下，可编辑文字项目是停用的，仅当用户单击“确认”按钮后，您才使用GetDialogItemText过程读这些项目中的信息

- 如果用户按下鼠标按钮的同时，光标在一个控制部件中，ModalDialog就调用控制管理器的函数TrackControl。如果用户释放鼠标按钮的同时，光标在一个有效的控制部件中，ModalDialog就返回控制部件的项目编号。您的应用程序应该适当地作出响应，例如在用户单击“确定”按钮后，执行一个命令

- 如果用户按下鼠标按钮的同时，光标在对话框的其它任何有效项目中，ModalDialog 就返回该项目的项目编号，您的应用程序应该适当地响应。在一般情况下，应该只响应控制功能。如果您的应用程序创建一个比按钮、选项按钮或复选框更复杂的控制部件，它 必须用您的事件过滤函数在该项目中处理事件

- 如果用户按下鼠标按钮的同时，光标在一个无效的项目或不在项目中，或者发生了其它任何事件，ModalDialog 不作任何事情

特殊考虑

不要对可移动定模式对话框（即以 movableDBoxProc 窗口定义标识符创建的窗口）或不定模式对话框（即以 noGrowDocProc 窗口定义标识符创建的窗口）使用 ModalDialog。如果您需要对话管理器帮助您处理可移动定模式对话框和不定模式对话框中的事件，应使用 IsDialogEvent 和 DialogSelect 函数。

ModalDialog 过程调用带有事件类型屏蔽参数的事件管理器函数GetNextEvent，该事件类型屏蔽参数排除了磁盘插入事件。若要接收磁盘插入事件，您的事件过滤函数可以调用事件管理器的过程 SetSystemEventMask。

当 ModalDialog 调用 TrackControl 时，它不允许您指定比按钮、选项按钮和复选框更复杂的动作过程。如果您需要更复杂的控制部件（例如，一个测量用户按住鼠标按钮的时间，或者测量用户把指示符移动的距离的控制部件），您可以创建自己的控制部件，或一个图片，或一个定义应用程序的项目，它在您的对话框中绘制类似于控制部件的对象。然后您必须提供一个事件过滤函数，它能够适当地处理该项目中的事件。

亦请参阅

6-83 页的列表 6-26 说明了 ModalDialog 的使用。6-79 页的“响应可编辑文字项目中的事件”一节说明了 ModalDialog 如何使用 TextEdit 处理文字输入和可编辑文字项目中的编辑。6-138 和 6-139 页说明了 IsDialogEvent 和 DialogSelect 函数（您的应用程序用它们来代替 ModalDialog 处理不定模式对话框和可移动定模式对话框）。有关您的应用程序应该在 filterProc 参数中指定的事件过滤函数的信息，请参阅 6-145 页中的 MyEventFilter。

本书的“事件管理器”一章对 GetNextEvent 和 SetSystemEventMask 作了说明。有关磁盘插入事件的讨论也请参阅这一章。有关您的应用程序应该如何响应控制部件中的事件的说明，请参阅 6-78 页的“响应控制部件中的事件”一节；TrackControl 函数在本书的“控制管理器”一章中作了详细的说明。有关创建您自己的非标准控制部件的信息，也请参阅这一章。TextEdit 在《Inside Macintosh 文本》*(Inside Macintosh: Text)* 一书的“TextEdit”一章中作了说明。

IsDialogEvent

若要在事件发生时，确定一个不定模式对话框或可移动定模式对话框是否有效，应使用 IsDialogEvent 函数。

```
FUNCTION IsDialogEvent  (theEvent： EventRecord) : Boolean ;
```

theEvent 一个事件记录，它是由事件管理器的函数，例如WaitNextEvent返回的。

说明

如果当您的对话框有效时，发生任何事件（包括空事件），IsDialogEvent 均返回 TRUE；否则，它返回 FALSE。当 IsDialogEvent 返回 FALSE 时，您应该把事件传送给事件处理代码的其余部分。当 IsDialogEvent 返回 TRUE 时，您应该在测试 DialogSelect 没有处理的事件后，把该事件传送给 DialogSelect。

一个对话记录包含窗口记录。当您使用 GetNewDialog、NewDialog 或 NewColorDialog 创建一个对话框时，对话管理器把窗口记录中的 windowKind 域设定为 dialogKind。IsDialogEvent 过程可通过 windowKind 域来判断对话框是否有效。

在把事件传送给 DialogSelect 前，如果 IsDialogEvent 返回的是 TRUE，您就应该执行以下的测试：

- 检查该事件是否是 Return 键、Enter 键、Esc 键或"命令"＋"."组合键的键按下事件。当用户按下 Return 键或 Enter 键时，您的应用程序应该和用户单击系统设定按钮一样响应；当用户按下 Esc 键或"命令"＋"."组合键时，您的应用程序应该和用户单击取消按钮一样响应。应使用 HiliteControl 过程把相应的按钮醒目显示 8 Tick 个时间
- 这时，您也许希望对那些您不准备传送给 DialogSelect 或在传送给 Dialog-Select 之前需要特殊处理的事件，进行检查和响应。例如，如果对话框需要响应磁盘插入事件，您就需要做这一步
- 检查该事件是否为一个非对话框的窗口的更新事件，如果是，就更新您的窗口
- 对于您创建的复杂项目，例如模拟复杂控制的图片或定义应用程序的项目，适当地测试并响应这些项目中的鼠标按下事件。当 DialogSelect 调用 TrackControl 时，它不允许您指定比按钮、选项按钮和复选框更复杂的动作过程。如果您需要更复杂的控制部件（例如，测量用户按住鼠标按钮的时间，或者测量用户把指示符移动的距离的一个控制部件），您可以创建自己的控制部件，或一个图片，或一个定义应用程序的项目，它在您的对话框中绘制类似于控制部件的对象。然后您必须自己测试并响应这些项目

如果当您显示一个可移动定模式对话框时，您的应用程序使用 IsDialogEvent 帮助处理事件，您在把事件传送给 DialogSelect 之前应该执行下列附加的测试：

- 测试可移动定模式对话框的标题栏中的鼠标按下事件，并通过相应地拖拽该对话框进行响应
- 测试并响应苹果菜单中的鼠标按下事件，如果可移动定模式对话框包括可编辑文字项目，还要测试并响应编辑菜单中的鼠标按下事件（当您显示一个可移动定模式对话框时，应该停用其它所有菜单。）
- 对于在可移动定模式对话框外的所有鼠标按下事件，播放系统示警声

特殊考虑

在 System 7 以前的系统软件版本中运行 IsDialogEvent 和 DialogSelect 都是不可靠的。如果您希望应用程序能够在较早的系统版本上运行，就不应该使用这些例程。

DialogSelect

在确定发生了和有效的不定模式对话框或可移动定模式对话框有关的事件后，您可以使用 DialogSelect 函数处理对话框中的大多数事件。

```
FUNCTION DialogSelect  (theEvent : EventRecord;
                        VAR theDialog : DialogPtr;
                        VAR itemHit : Integer) : Boolean;
```

theEvent 一个事件记录，它是由事件管理器的函数，例如 WaitNextEvent 返回的。

theDialog 对话框的对话记录的一个指针，事件发生在该对话框中。

itemHit 项目编号，它代表选定的项目在有效对话框的项目列表资源中的位置。

说明

DialogSelect 函数处理和对话框有关的大多数事件。如果该事件是对话框的一个激活或更新事件，DialogSelect 就激活或更新对话框，并返回 FALSE。如果该事件涉及一个有效项目，DialogSelect 返回 TRUE 作为函数结果。它通过 itemHit 参数中返回用户选定的项目的项目编号。在 theDialog 参数中，DialogSelect 返回事件在其中发生的对话框的对话记录的一个指针。在其它所有的情况下，DialogSelect 返回 FALSE。当 DialogSelect 返回 TRUE 时，您应对涉及这个特定对话框中用户选定项目的事件作出适当的响应；当返回的是 FALSE 时，您不需要作任何事情。

在一般情况下，对话框中只能启用控制部件；因此您的应用程序应该只有在用户单击一个启用的（即有效的）控制部件，例如“确认”按钮后，DialogSelect 返回 TRUE 时，才进行响应。

DialogSelect 函数首先获得包含该事件的窗口的一个指针。对于更新和激活事件，事件记录包含窗口的指针。对于其它类型的事件，DialogSelect 调用窗口管理器的函数 FrontWindow。然后，对话管理器通过调用 QuickDraw 过程 SetPort 使这个窗口成为当前图形端口。如果在有效对话框中有任何可编辑文字项目，DialogSelect 就设置文字信息，准备处理该事件。

如果该事件是对话框的一个更新事件，DialogSelect 依次调用窗口管理器的过程 BeginUpdate、对话管理器的过程 DrawDialog 和窗口管理器的过程 EndUpdate。当项目是控制（‘CNTL’）资源中定义的一个控制部件时，添加至更新区域的矩形是在控制资源中定义的矩形，而不是在项目列表资源中定义的显示矩形。

DialogSelect 函数对事件作如下处理:

- DialogSelect 响应对话框中的激活或更新事件，激活或更新它的窗口，并返回 FALSE
- 如果发生一个键按下事件或自动键事件，并且对话框中有一个可编辑文字项目，DialogSelect 就使用 TextEdit 处理文字输入和编辑，并返回 TRUE 作为函数结果。DialogSelect 在自己的 itemHit 参数中返回项目编号
- 如果发生一个键按下事件或自动键事件，但对话框中没有可编辑文字项目，DialogSelect 返回 FALSE
- 如果用户按下鼠标按钮的同时，光标在一个可编辑文字项目中，DialogSelect 就适当地响应鼠标活动，即显示一个插入点或者选定域中内容。如果可编辑文字项目是无效的，DialogSelect 就返回 FALSE。如果可编辑文字项目被启用，DialogSelect 就返回 TRUE，并在自己的 itemHit 参数中返回项目编号。在正常情况下，可编辑文字项目是无效的，仅当用户单击“确认”按钮后，您才使用 GetDialogItemText 过程读取项目中的信息

- 如果用户按下鼠标按钮的同时，光标在一个控制部件中，DialogSelect调用控制管理器的函数TrackControl。如果用户释放鼠标按钮的同时，光标在一个有效的控制部件中DialogSelect返回TRUE作为函数结果，并在自己的itemHit参数中返回控制部件的项目编号。您的应用程序应该适当地作出响应，例如在用户单击"确认"按钮后，执行一个命令

- 如果用户按下鼠标按钮的同时，光标在对话框的其它任何有效项目中，DialogSelect返回TRUE作为函数结果，并在自己的itemHit参数中返回该项目的项目编号，您的应用程序应该适当地响应。在一般情况下，应该只启用控制部件。如果您的应用程序创建一个复杂的控制部件，例如测量一个拨号移动了多远，您的应用程序应该先处理该项目中的鼠标事件，再把事件传送给DialogSelect

- 如果用户按下鼠标按钮的同时，光标在一个无效的项目或不在项目中，或者发生了其它任何事件，DialogSelect不作任何事情

- 如果该事件不是DialogSelect要处理的事件（例如是一个空事件），而在对话框中有一个可编辑文字项目，DialogSelect就调用TextEdit的过程TEIdle 使插入点闪烁

特殊考虑

因为DialogSelect只处理对话框中的鼠标按下事件和对话框的可编辑文字项目中的键按下事件，您应该适当地处理其它事件，再把它们传送给DialogSelect。同样，当DialogSelect调用TrackControl时，它不允许您指定比按钮、选项按钮和复选框更复杂的动作过程。如果您需要更复杂的控制部件（例如，一个测量用户按住鼠标按钮的时间，或者测量用户把指示符移动的距离的控制部件），您可以创建自己的控制部件，或一个图片，或一个定义应用程序的项目，它在您的对话框中绘制类似于控制部件的对象。然后您必须自己测试并响应这些事件。

在对话框中，应使用过程DialogCut，DialogCopy，DialogPaste和DialogDelete支持可编辑文字方框中的"剪切"、"复制"、"贴"和"清除"命令。

在 System 7 以前的系统软件版本中运行DialogSelect是不可靠的。如果您预期到应用程序会在较早的系统版本上运行，就不应该使用这个例程。

亦请参阅

6-79 页的列表 6-25 解释了在只有空事件时，调用DialogSelect使可编辑文字项目中的光标闪烁的方法；6-92 页的程序清单6-29 给出了使用DialogSelect处理不定模式对话框中的鼠标按下事件的实例；6-96 页的列表 6-33 给出了使用DialogSelect 处理可编辑文字项目中的键按下事件的实例；6-98 页的程序清单 6-34给出了使用DialogSelect 处理不定模式对话框中的激活事件的实例。

DrawDialog

如果您不使用其它任何对话管理器例程处理对话框中的事件，您可以使用 DrawDialog 过程绘制它的全部内容。

```
PROCEDURE DrawDialog (theDialog : DialogPtr) ;
```

theDialog　　对话记录的一个指针。

说明

DrawDialog 过程绘制指定对话框中的全部内容。DrawDialog 过程绘制所有的对话项目，调用控制管理器过程 DrawControl 绘制所有的控制部件，并调用 TextEdit 的过程 TEUpdate 更新所有的静态和可编辑文字项目并绘制它们的显示矩形。如果项目的矩形在更新区域中，DrawDialog 过程还调用应用程序定义的项目绘制过程。

DialogSelect，ModalDialog，Alert，StopAlert，NoteAlert 及 CautionAlert 例程自动使用 DrawDialog。如果您使用 GetNewDialog 创建一个对话框，但是在处理对话框中的事件时，没有使用对话管理器的其它任何例程，您可以在对话框可见时，使用 DrawDialog 重新绘制它的内容。如果对话框不可见，应首先使用窗口管理器的过程 ShowWindow，然后使用 DrawDialog。

亦请参阅

有关窗口中的激活和更新事件更详细的信息，请参阅本书的“窗口管理器”和“事件管理器”两章。本书的“控制管理器”一章中对 DrawControl 过程作了说明。有关 TEUpdate 过程请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书的“TextEdit”一章。

UpdateDialog

您可以使用 UpdateDialog 过程重新绘制指定对话框的更新区域。UpdateDialog 过程和 UpdtDialog 过程是一样的。

```
PROCEDURE UpdateDialog (theDialog : DialogPtr ;
                        updateRgn : RgnHandle) ;
```

theDialog　　对话记录的一个指针。

updateRgn　　需要更新的窗口区域的一个间接指针。

说明

UpdateDialog过程仅重新绘制updateRgn参数中指定的对话框中的区域。由于DialogSelect，ModalDialog，Alert，StopAlert，NoteAlert和Caution-Alert例程都自动调用DrawDialog处理您的警告栏和对话框中的更新事件，您的应用程序可能永远不会需要使用UpdateDialog。

UpdateDialog不是绘制指定对话框中的全部内容，而仅绘制指定更新区域中的项目。您可以使用UpdateDialog响应一个更新事件，通常您应该把它夹在窗口管理器的过程 BeginUpdate 和EndUpdate之间。UpdateDialog过程使用QuickDraw的过程SetPort把对话框变成当前图形端口。UpdateDialog使用控制管理器的过程UpdateControl来绘制控制部件，它比DrawControl过程绘制得快。

亦请参阅

6-99 页的列表 6-35 给出了使用UpdateDialog响应不定模式对话框中的更新事件的实例。有关窗口的激活和更新事件更详细的信息，请参阅本书的“窗口管理器”一章。UpdateControl过程在本书的“控制管理器”一章中说明。

应用程序定义的例程

如果您在对话框中提供了一个自己定义的项目，就必须为对话管理器提供一个绘制过程，以便在显示该项目时调用。这个过程在本节中被命名为MyItem。如果您希望对话管理器发出的语音不是系统示警声，就必须提供您自己的示警声过程，它在本节中被命名为MyAlertSound。若要补充对话管理器在Macintosh 多任务环境中处理事件的能力，您应该提供一个事件过滤函数，供对话管理器在显示警告栏和定模式对话框时调用。这个函数在这里被命名为 MyEventFilter。

MyItem

若要在对话框中绘制您自己定义的项目，应提供一个绘制过程，它接受两个参数：对话框的一个窗口指针和在对话框项目列表资源中的项目编号。如下面为MyItem过程说明：

```
PROCEDURE MyItem (theWindow: WindowPtr; itemNo: Integer);
```

theWindow 对话框的对话记录的一个指针，该对话框包含一个定义应用程序的项目。如果您的过程可以在一个以上的对话框中进行绘制，这个参数指示在哪个对话框中绘制。

itemNo

项目编号，它相应于一个项目在指定对话框的项目列表资源中的位置。如果您的过程能够绘制一个以上的项目，这个参数告诉您的过程应该绘制哪一个项目。

说明

在您显示指定对话框时，对话管理器调用您的过程，绘制一个应用程序定义的项目。当调用您的绘制过程时，对话管理器把当前端口设定为该对话框的图形端口。在正常情况下，您创建的是一个不可见对话框，然后调用窗口管理器过程 ShowWindow 显示这个对话框。

在您显示这个对话框前，应使用 SetDialogItem 过程把这个过程安装在对话记录中。您在使用 SetDialogItem 之前，必须先使用 GetDialogItem 过程获得一个 userItem 类型（用户自定义类型）的项目的间接指针。

如果您使用该过程绘制的这个项目被您启用，当用户单击该项目时，ModalDialog 和 DialogSelect 函数就返回它的项目编号。如果您的应用程序需要响应更为复杂的用户动作(例如，如果您的应用程序需要测量用户按住鼠标按钮的时间或者用户拖拽光标的距离)，它必须自己跟踪光标。如果您使用 ModalDialog，您的事件过滤函数就必须处理该项目中的事件；如果您使用 DialogSelect，您的应用程序在把事件传送给 DialogSelect 之前，必须处理该项目中的事件。

亦请参阅

6-59 页的列表 6-17 给出了一个绘制任意形状和大小的按钮的黑体轮廓线的过程；6-58 页的列表 6-16 说明了使用 GetDialogItem 和 SetDialogItem 在对话记录中安装这个绘制过程的方法。有关 ShowWindow 过程请参阅本书的“窗口管理器”一章。

MyAlertSound

如果您希望窗口管理器发出的语音不是系统示警声，应编写自己的示警声过程，并调用 ErrorSound 过程使您的示警声过程成为当前示警声过程。如果您命名这个过程为 MyAlertSound，就可以这样宣布它：

```
PROCEDURE MyAlertSound  (soundNo : Integer) ;
```

soundNo　从 0 到 3 的一个整数，它代表可能的 4 级警告。

说明

对于可以在soundNo参数中报告的 4 个警告级中的任何一个，您的过程都可以发出您定义的任何示警声。当对话管理器调用您的过程时，它把 0 作为一个示警声的语音编号传送，这个示警声是由警告资源中的silent常数指定的。对话管理器为sound1常数指定的语音传送 1，为sound2常数指定的语音传送 2，为sound3常数指定的语音传送 3。

特殊考虑

当对话管理器检测到在警告栏或定模式对话框外面单击鼠标的事件时，它使用声音管理器过程SysBeep发出系统示警声。通过改变声音控制板的设定状态，用户可以选定一种系统示警声。为了和系统软件及其它Macintosh应用程序保持一致，每当您的语音过程收到语音编号 1 （您可以用sound1常数代表）时，应该调用SysBeep。

亦请参阅

6-22 页的列表 6-3 说明了如何使用MyAlertSound过程。有关SysBeep过程请参阅 <<Inside Macintosh声音>> *(Inside Macintosh : Sound)* 一书。

MyEventFilter

若要补充对话管理器处理事件的能力，您的应用程序应该提供一个事件过滤函数，供对话管理器在显示警告栏和定模式对话框时调用。您的事件过滤函数应该有 3 个参数，并送回布尔值。下面为MyEventFilter过程说明：

```
FUNCTION MyEventFilter (theDialog: DialogPtr;
                        VAR theEvent: EventRecord;
                        VAR itemHit: Integer): Boolean;
```

theDialog 警告栏或定模式对话框的对话记录的一个指针。

theEvent 由事件管理器的函数，例如WaitNextEvent返回的一个事件记录。

itemHit 项目编号，它相应于一个项目在警告栏或定模式对话框的项目列表资源中的位置。

您的函数在收到它不处理的事件后，应返回 FALSE。当您的函数返回 FALSE 时，ModalDialog 处理您在参数 theEvent 中传送的这个事件。(为了模拟一个不同的事件，您的函数也可以改变该事件并返回 FALSE，您的函数把这个事件传送给对话管理器处理。) 如果您的函数确实能够处理该事件，就应该返回 TRUE 作为函数结果，并在 itemHit 参数中返回它处理的对话项目的编号。然后，ModalDialog 过程和 Alert，NoteAlert，StopAlert 和 CautionAlert 函数都在它们自己的 itemHit 参数中返回这个项目编号。

您的事件过滤函数应该执行下列任务：

- 若用户按下 Return 键或 Enter 键，应返回 TRUE 和系统设定按钮的项目编号
- 若用户按下 Esc 键或“命令”＋“.”组合键，应返回 TRUE 和取消按钮的项目编号
- 响应更新事件更新您的窗口(这使得后台应用程序能够接收更新事件)并返回FALSE
- 对于您的事件过滤函数不处理的所有事件返回 FALSE

您也可以使用事件过滤函数测试和响应等效命令组合键和更复杂的事件，比如用户在定义应用程序的项目中拖拽光标。如果您提供一个定义应用程序的项目，它要求您测量用户按住鼠标按钮的时间或者用户拖拽光标的距离，就应该使用事件过滤函数处理该项目中的事件。

ModalDialog 过程调用带有事件类型屏蔽参数的事件管理器函数 GetNext-Event，该事件类型屏蔽参数排除磁盘插入事件；若要接收磁盘插入事件，您的事件过滤函数可以调用事件管理器过程 SetSystemEventMask。

您可以在大多数或所有的警告栏和对话框中使用同样的事件过滤函数。

对于警告栏和定模式对话框，对话管理器提供了一个标准事件过滤函数，它检查用户是否按下了 Enter 键或 Return 键，如果按了，就返回系统设定按钮的项目编号。您的事件过滤函数应该总是检查 Return 键和 Enter 键是否被按下，如果被按下，应在 itemHit 参数中返回系统设定按钮的项目编号，并返回 TRUE 作为函数结果。

在所有的警告栏和对话框中，被键按下事件激活的任何按钮都应该反色，以便显示选定的项目。应使用控制管理器的过程 HiliteControl 把一个按钮反色 8 个 Tick 时间，这个时间足以使用户注意，又不是太长而令人厌烦。当用户单击按钮时控制管理器执行这个动作，当用户使用单击按钮的等效命令组合键时，您的应用程序应该完成这个任务。

对于包含可编辑文字项目的定模式对话框，您的应用程序应该处理菜单栏的存取，允许用户使用您的编辑菜单和它的“剪”、“复制”、“贴”、“清除”和“取消”命令。然后，您的事件过滤函数应该测试并处理编辑菜单中的点按鼠标事件及适当命令的等效命令组合键事件。您的应用程序应该使用DialogCut、DialogCopy、DialogPaste和DialogDelete过程支持“剪”、“复制”、“贴”和““清除””命令。

对于警告栏，您应在传送给Alert、StopAlert、CautionAlert和NoteAlert函数的参数中指定您的事件过滤函数的一个指针。对于定模式对话框，应在您传送给ModalDialog过程的参数中指定您的事件过滤函数的一个指针。

亦请参阅

6-88 页的列表 6-27 给出了一个事件过滤函数。有关函数GetNextEvent和SetSystemEventMask请参阅本书的“事件管理器”一章。

资源

本节介绍对话管理器为显示警告栏和对话框而使用的资源。这些资源是：

- 对话（'DLOG'）资源，它指定了窗口类型、显示矩形和警告栏的项目列表资源
- 警告（'ALRT'）资源，它指定了示警声、显示矩形和警告栏的项目列表资源
- 项目列表（'DITL'）资源，它指定了在警告栏或对话框中显示的项目，例如按钮和静态文字
- 对话项目色彩表（'dctb'）资源，它使您为对话框提供一个彩色图形端口，并能在对话框中使用系统设定色彩以外的色彩
- 警告色彩表（'actb'）资源，它使您能在警告栏中使用系统设定色彩以外的色彩
- 项目色彩表（'ictb'）资源，它使您能够在警告栏或对话框中改变项目的系统设定色彩、字体、字形和字体大小

本节介绍了这些资源在用Rez资源编译程序编译后的结构，Rez资源编译程序可从APDA获得。如果您有兴趣为这些资源创建Rez输入文件，请参阅 6-17 页的“使用对话管理器”一节中的详细信息。

对话资源

您可以使用对话资源定义一个对话框。对话资源是一个 'DLOG' 类型的资源。所有的对话资源必须标记为可清除的，它们的资源标识符必须大于 128。

为了在对话框中指定项目，您还必须提供一个项目列表资源，6-151 页对此作了说明。应使用 GetNewDialog 函数（请参阅 6-113 页）创建在对话资源中定义的对话框。

对话资源的 Rez 输入文件的格式和它编译后的输出格式不同。本节介绍 Rez 编译后的对话资源。如果您只关心创建一个对话资源，请参阅 6-23 页开始的“创建对话框”一节。

图 6-42 给出了编译了的对话资源的格式。

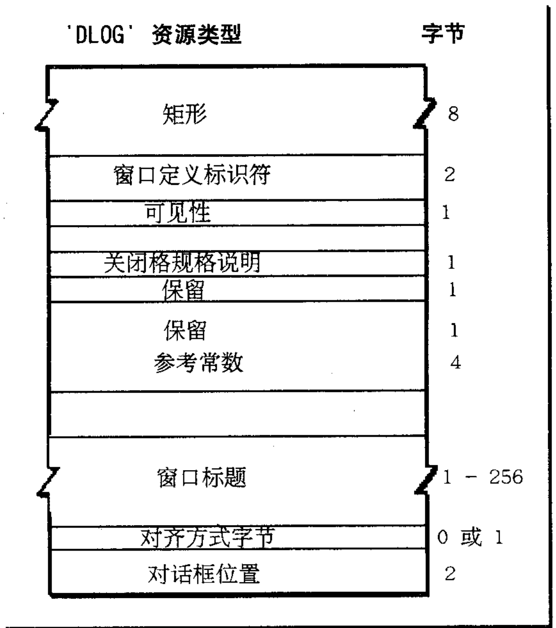


图 6-42 编译了的对话（'DLOG'）资源的结构

编译后的对话资源包含以下元素：

- 矩形。它决定了对话框的尺寸，也可能决定了它的位置（对话资源中的最后一个元素通常指定了对话框的一个位置）
- 窗口定义标识符
- 如果这里出现整数 0（由 dBoxProc 窗口定义标识符 在 Rez 输入文件中指定），对话管理器就显示一个定模式对话框

- 如果整数 4 在这里出现（由noGrowDocProc窗口定义标识符 在Rez输入文件中指定），对话管理器就显示一个不定模式对话框。
- 如果整数 5 在这里出现（由movableDBoxProc窗口定义标识符 在Rez输入文件中指定），对话管理器就显示一个可移动定模式对话框。

这些类型的对话框分别在 6-10 页的图 6-6、6-12页的图 6-8 和 6-11 页的图 6-7 中作了说明。

- 可见性。如果这个元素设定为 1（由Rez输入文件的visible常数指定），对话管理器在您一调用GetNewDialog时就显示对话框。如果它被设定为 0（由Rez输入文件的invisible常数指定），对话管理器在您调用窗口管理器的过程ShowWindow之前，不显示对话框

- 关闭格规格说明。它指定了是否绘制关闭格。在正常情况下，只有在不定模式对话框的标题栏上指定一个关闭格时，它被设定为 1（由Rez输入文件的goAway常数指定），否则总是设定为 0（由Rez输入文件的noGoAway常数指定）

- 参考常数。这个元素包含了应用程序储存在这里的任何值。例如，一个应用程序可以在这里储存一个代表对话框类型的数字，或者储存一个记录的间接指针，该记录保留对话框或其它窗口类型的状态信息。应用程序可以在任何时候使用窗口管理器过程SetWRefCon在对话框的对话记录中改变这个值，您也可以使用GetWRefCon函数确定它的当前值

- 项目列表资源标识符。该项目列表资源指定了在对话框中显示的项目，例如按钮和静态文字

- 窗口标题。这是一个Pascal字符串，只有当对话框是一个不定模式对话框时，才在对话框的标题栏上显示它

- 对齐方式字节。这是一个额外的字节，必要时才添加。它使上面的Pascal字符串在一个字边界结束

- 对话框位置。这个元素指定了对话框在屏幕上的位置（如果您的应用程序自己为对话框定位，请不要使用这些常数，因为您的代码可能和对话管理器冲突。）
  - 如果 0×0000 在这里出现（由Rez输入文件的noAutoCenter常数指定），对话管理器把这个对话框按照资源的矩形元素中指定的全局坐标定位。
  - 如果 0×B00A 在这里出现（由Rez输入文件的alertPositionParentWindow 常数指定），对话管理器把这个对话框定位在最前面的窗口，使窗口标题出现。请 参阅6-63 页的图 6-33。
  - 如果 0×300A 在这里出现（由Rez输入文件的alertPositionMainScreen常数指定），对话管理器在靠近主屏幕的顶部把这个对话框置中。请参阅 6-63 页的图 6-34。
  - 如果 0×700A 在这里出现（由Rez输入文件的alertPositionParentWindowScreen常数指定），对话管理器把对话框放置在用户当前工作的区域。请参阅 6-64 页的图 6-35。

警告资源

您可以使用警告资源定义一个警告。警告资源是一个 'ALRT' 类型的资源。所有警告资源必须标记为可清除的，它们的资源标识符 必须大于 128。

若要在一个警告栏中指定项目，您还必须提供项目列表资源，6-151 页将有详细说明。为了显示警告信号，您应调用 NoteAlert，CautionAlert，StopAlert 或 Alert 函数，并把警告资源的资源标识符 传送给它。6-105 页的“创建警告信号”中对于 NoteAlert，CautionAlert，StopAlert 和 Alert 函数作了说明。

警告资源的 Rez 输入文件格式和它的编译了的输出格式不同。本节说明用 Rez 编译了的警告资源的结构。如果您只关心创建警告资源，请参阅 6-18 页的“创建示警声和警告栏”一节。

图 6-43 给出编译了的警告资源的结构。

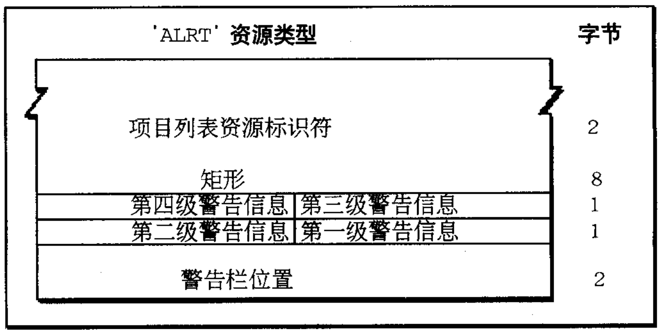


图 6-43 编译了的警告（ 'ALRT' ）资源

编译后的警告资源包含下列元素：

- 矩形。它决定了对话框的尺寸，也可能决定了它的位置（警告资源中的最后一个元素通常指定了对话框的一个位置。）
- 项目列表资源标识符。该项目列表资源指定了在对话框中显示的项目，例如按钮和静态文字
- 第四级警告信息。它指定了当用户连续重复请求这个警告信号 4 次或 4 次以上时的响应。对话管理器按照在这个参数指定的状态进行响应
  - 如果第一位被设定，对话管理器就环绕项目列表资源中的第二个项目绘制一个黑体轮廓线（通常是“取消”按钮），当用户在警告信号连续第四次出现时按下 Return 键或 Enter 键，而您的应用程序不指定事件过滤函数，就返回 2。如果第一位没有设定，对话管理器就环绕项目列表资源中的第一个项目绘制一个黑体轮廓线（通常是“确认”按钮），当用户按下 Return 键或 Enter 键时 ，而您的应用程序不指定事件过滤函数，就返回 1。

- 如果第二位被设定，对话管理器就在这一级显示警告栏。如果第二位没有设定，对话管理器在这一级不显示警告栏。
- 如果下两位都未设定，对话管理器在这一级就不发出示警声。如果第三位被设定，第四位未设定，对话管理器就发出第一示警声，它的系统设定值是系统示警声。如果第三位未设定，第四位被设定，对话管理器就播放第二警告语音，它的系统设定值是播放系统示警声两次。如果第三和第四位都被设定，对话管理器就播放第三示警声，它的系统设定值是播放系统示警声三次。通过定义自己的示警声 请参阅 6-144 页）并调用 ErrorSound 过程 请参阅 6-104 页）使您的示警声成为当前示警声过程，您可以指定自己的示警声。

- 第三级警告信息。它指定了当用户连续重复请求这个警告信号 3 次时的响应。对话管理器对它的 4 个数位的解释和第四级警告信息一样

- 第二级警告信息。它指定了当用户连续重复请求这个警告信号 2 次时的响应。对话管理器对它的 4 个数位的解释和第四级警告信息一样

- 第一级警告信息。它指定了当用户第一次执行请求这个警告信号的动作时的响应。对话管理器对它的 4 个数位的解释和第四级警告信息一样

- 警告栏位置。这个元素指定了警告栏在屏幕上的位置 如果您的应用程序自己为警告栏定位，请不要使用这些常数，因为您的代码可能和对话管理器冲突。）
  - 如果 0 × 0000 在这里出现 由 Rez 输入文件的 noAutoCenter 常数指定），对话管理器把这个警告栏按照资源的矩形元素中指定的全局坐标定位。
  - 如果 0 × B00A 在这里出现 由 Rez 输入文件的 alertPositionParentWindow 常数指定），对话管理器把这个警告栏定位在最前面的窗口，使窗口标题出现。请参阅 6-63 页的图 6-33。
  - 如果 0 × 300A 在这里出现 由 Rez 输入文件的 alertPositionMainScreen 常数 指定），对话管理器在靠近主屏幕的顶部把这个警告栏置中。请参阅 6-63 页的图 6-34。
  - 如果 0 × 700A 在这里出现 由 Rez 输入文件的 alertPositionParentWindowScreen 常数指定），对话管理器把警告栏放置在用户当前工作的区域。请参阅 6-64 页的图 6-35。

项目列表资源

您使用项目列表资源在警告栏和对话框中指定项目，例如按钮和文字。项目列表资源是一个 DITL’ 类型的资源。所有项目列表资源必须标记为可清除的，它们的资源标识符 必须大于 128。

对于一个警告栏，您在警告资源（请参阅 6-150 页的说明）中指定项目列表资源的资源标识符。对于您用GetNewDialog创建的对话框，您在对话资源（请参阅6-148 页的说明）中指定项目列表资源的资源标识符。对于您用NewColorDialog函数（请参阅 6-115 页的说明）或NewDialog函数（请参阅 6-118 页的说明）创建的对话框，您使用资源管理器函数GetResource把项目列表资源读入存储器，并在存储器中提供项目列表资源的一个间接指针。

项目列表资源的Rez输入文件格式和它的编译了的输出格式不同。本节介绍用 Rez 编译了的警告资源的结构。如果您只关心创建项目列表资源，请参阅 6-26 页的“为警告栏和对话框提供项目”一节

图 6-44 给出了编译了的项目列表资源的结构。

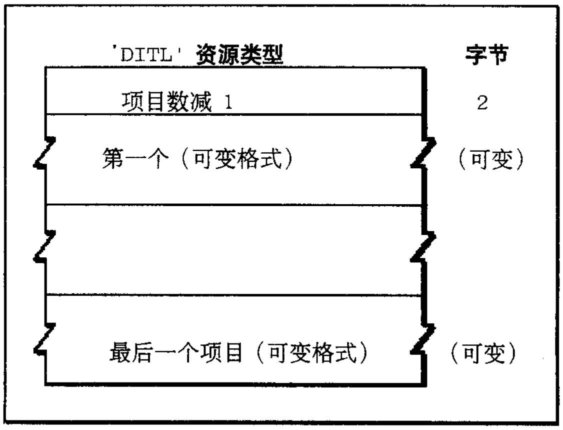


图 6-44 编译后的项目列表（ 'DITL' ）资源

编译后的项目列表资源包含下列元素：

- 项目数减 1。这个值比该资源中定义的项目总数少 1
- 一个可变的项目数

每个项目的格式取决于它的类型。图 6-45 给出了定义为按钮、复选框、选项按钮、静态文本项目，或可编辑文字项目的格式。

编译了的按钮、复选框、选项按钮、静态文本项目，或可编辑文字项目包含下列元素：

- 保留。对话管理器把这个元素用于储存
- 显示矩形。它决定了警告栏或对话框中项目的大小和位置。显示矩形用警告栏或对话框的局部坐标指定；这些坐标指定了该项目的左上角和右下角

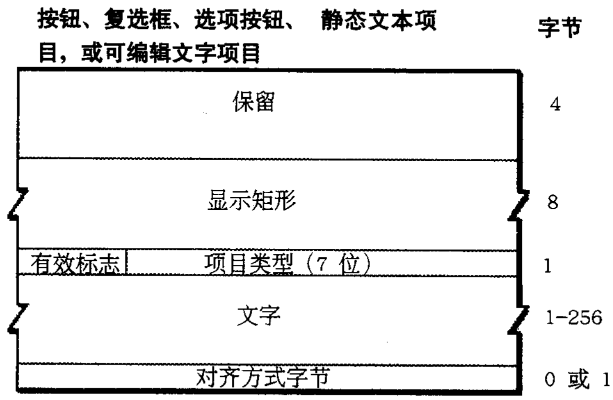


图 6-45 编译了的按钮、复选框、选项按钮、静态文本项目或可编辑文字项目的结构

- 有效标志。它指定了项目是否有效。如果这个数位被设定，该项目是有效的，每当鼠标按下事件在这个项目中发生时，对话管理器就向您的应用程序报告

- 项目类型
  · 如果这个位串被设定为 4（由 Rez 输入文件的 Button 常数指定），该项目是一个按钮。
  · 如果这个位串被设定为 5（由 Rez 输入文件的 CheckBox 常数指定），该项目是一个复选框。
  · 如果这个位串被设定为 6（由 Rez 输入文件的 RadioButton 常数指定），该项目是一个选项按钮。
  · 如果这个位串被设定为 8（由 Rez 输入文件的 StaticText 常数指定），该项目是一个静态文字。
  · 如果这个位串被设定为 16（由 Rez 输入文件的 EditText 常数指定），该项目是一个可编辑文字。

- 文字。它指定了出现在项目中的文字。这个元素由一个字符串长度位和另外 255 个字符串组成（6-37 页的“按钮、复选框和选项按钮”和 6-40 页的“静态文字和可编辑文字项目的文字字符串”两节中推荐了用于项目的适当文字。）
  · 对于按钮、复选框或选项按钮，它是该控制部件的标题。
  · 对于静态文本项目，它是该项目的文字。
  · 对于可编辑文字项目，它可以是一个空字符串（这时，可编辑文字项目不包含文字），它也可以是一个字符串，这个字符串作为可编辑文字项目的系统设定字符串出现。

- 对齐方式字节。如果必要就添加，它使上面的文字字符串在字的边界结束

图 6-46 给出了定义为控制部件、图标或图片项目的元素的格式。

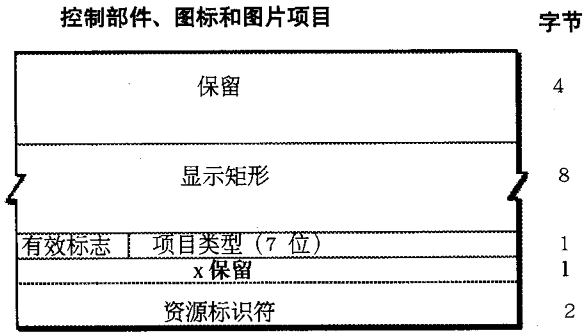


图 6-46 编译后的控制部件、图标和图片项目的结构

编译后的控制部件、图标或图片项目包含下列元素：

- 保留。对话管理器把这个元素用于储存
- 显示矩形。它决定了警告栏或对话框中项目的大小和位置。显示矩形用警告栏或对话框的局部坐标指定
- 有效标志。它指定了该项目是否有效。如果这个数位被设定，该项目是有效的，每当鼠标按下事件在这个项目中发生时，对话管理器就向您的应用程序报告
- 项目类型
  - · 如果这个 7 位字符串被设定为 7（由 Rez 输入文件的 Control 常数指定），该项目是一个按钮。
  - · 如果这个 7 位字符串被设定为 32（由 Rez 输入文件的 Icon 常数指定），该项目是一个图标。
  - · 如果这个 7 位字符串被设定为 64（由 Rez 输入文件的 Picture 常数指定），该项目是一个 QuickDraw 图片。
- 资源标识符
  - · 对于一个控制部件项目，它是 ‘CTRL’ 资源的资源标识符。
  - · 对于一个图标项目，它是‘ICON’资源的资源标识符，也可选为 'cicn' 资源的资源标识符。
  - · 对于一个图片，它是 ‘PICT’ 资源的资源标识符。

图 6-47 给出了一个定义应用程序的项目的格式。

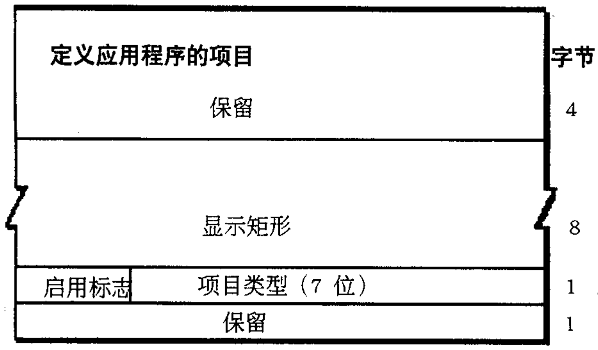


图 6-47 编译后的定义应用程序的项目的结构

编译了的定义应用程序的项目包含下列元素：

- 保留。对话管理器把这个元素用于储存
- 显示矩形。它决定了警告栏或对话框中定义应用程序的项目的大小和位置。显示矩形用警告栏或对话框的局部坐标指定
- 有效标志。它指定了该项目是否有效。如果这个数位被设定，该项目是有效的，每当鼠标按下事件在这个项目中发生时，对话管理器就向您的应用程序报告
- 项目类型。它的值被设定为 0（由 Rez 输入文件的 UserItem 常数指定）

图 6-48 给出了一个帮助项目的格式。[帮助项目在《Inside Macintosh 工具箱扩展》*(Inside Macintosh : More Macintosh Toolbox)* 一书的“帮助管理器”一章中有详细的说明。]

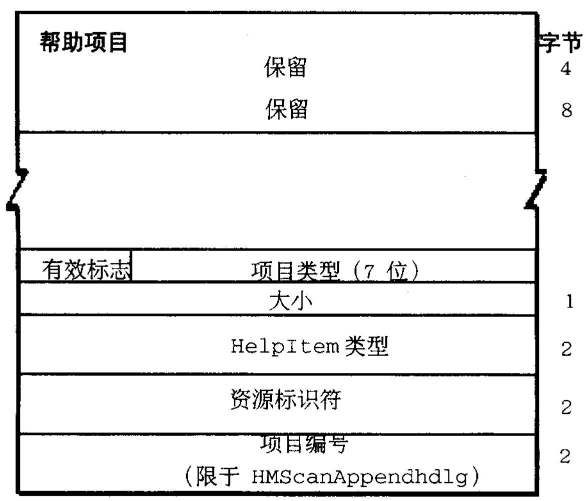


图 6-48 编译了帮助项目的结构

编译后的帮助项目包含下列元素：

- 保留。对话管理器把这个元素用于储存
- 保留。它应该被设定为 0
- 有效标志。它指定了该项目是否有效。对于帮助项目，这个项目不应该被设定。因为当鼠标按下事件在这个项目中发生时，对话管理器不必向您的应用程序报告
- 项目类型。它的值被设定为 1（由 Rez 输入文件的 HelpItem 常数指定）
- 大小。它指定了这个元素的其余部分包含的字节数。对于 HMScanhdlg 和 HMScan-hrct 标识符标识的项目，它被指定为 4，对于 HMScanAppendhdlg 标识符标识的项目，它被指定为 6
- HelpItem 类型。它指定了在资源中定义的帮助项目的类型
  - 对于 HMScanhdlg 标识符标识的项目，这个元素包含值 1。
  - 对于 HMScanhrct 标识符标识的项目，这个元素包含值 2。
  - 对于 HMScanAppendhdlg 标识符标识的项目，这个元素包含值 8。
- 资源标识符。它是包含这个警告栏或对话框的帮助消息的资源的资源标识符
  - 对于 HMScanhdlg 或 HMScanhrct 标识符标识的项目，它是 'hdlg' 资源的资源标识符。
  - 对于 HMScanAppendhdlg 标识符标识的项目，它是 'hrct' 资源的资源标识符。
- 项目编号。它只能用于 HMScanAppendhdlg 标识符标识的项目。这是在警告栏和对话框中的项目编号，'hdlg' 资源中指定的帮助消息在它的后面显示。这些帮助消息和增补在警告栏和对话框中的项目有关（对于 HMScanhdlg 或 HMScanhrct 标识符标识的项目，项目列表资源不包含这两个字节。）

对话项目色彩表资源

在彩色显示器上，对话管理器自动地为您的警告栏和对话框添加彩色，使得它们和系统软件使用的窗口、警告栏和对话框的色彩匹配。这些彩色为从黑白显示到 8 位彩色显示的所有显示器提供了美学上的一致性。例如，在一个彩色显示器上，不定模式对话框标题栏的跑道线是灰色，关闭格和窗口框架是彩色的，按钮和文字是黑色的。

当您创建对话资源时，您的应用程序的对话框使用系统设定色彩。通常您只需要为您的对话框提供色彩，但下列情况例外：

- 当您需要把一个图标的彩色版包括在对话框中时，必须创建一个 'cicn' 类型的资源，它和项目列表资源中指定的黑白 'ICON' 资源有相同的资源标识符。本书封面的版面 2 显示了一个包括彩色图标的警告栏

- 当您需要为绘制无效（即色彩暗淡）的系统设定按钮产生一种调和灰色时，您必须创建一个对话项目色彩表（ 'dctb' ）资源，它和对话资源有同样的资源标识符

6-56 页的“使用定义应用程序的项目绘制系统设定按钮的黑体轮廓线”一节解释了如何创建一个绘图例程，它绘制出对话框的系统设定按钮。如果您停用一个对话框，就应该使它的按钮的色彩变得暗淡，并使用灰色绘制系统设定按钮的轮廓线。因为 GetNewDialog 和 NewDialog 过程都为对话框提供黑白图形端口，您可以为对话框创建一个对话项目色彩表资源，强制对话管理器提供彩色图形端口。然后，您可以把调和灰色用于系统设定按钮的轮廓线。(NewColorDialog函数提供一个彩色图形端口。)

即使当您为绘制灰色轮廓线创建一个对话项目色彩表资源时，您也不应该改变系统的系统设定色彩。如果您感到一定要使用非标准的色彩，可以使用对话管理器指定系统设定色彩以外的色彩。您的应用程序可以通过创建一个对话项目色彩表（'dctb'）资源为对话框指定自己的色彩，这个资源必须和对话资源有同样的资源标识符（请参阅 6-148 页开始的说明）。您不需要调用任何新的例程改变对话框所使用的色彩。例如，当您调用 GetNewDialog 函数时，对话管理器自动地试图装入一个和对话资源有同样资源标识符 的对话项目色彩表资源。

但是，必须注意，您的对话框中的非标准色彩可能在一开始使用户感到迷惑。还应注意，无论您作了什么改变，用户总是可以通过改变彩色控制板上的设定值改变对话框的色彩。

警告:

因为彩色警告栏、彩色对话框、彩色项目和彩色图标在运行 System 7 以前的系统软件的电脑上是不可靠的，如果您希望保留和以前系统的兼容性，请不要创建这些色彩。

对话项目色彩表资源和窗口色彩表（即 'wctb' 类型的资源）的格式完全相同，后者的说明在本书的“窗口管理器”一章中给出。

如果对话框的内容色彩不是白色，应在对话资源中指定 invisible 常数。当对话框是最前面的窗口时，应使用窗口管理器的过程 ShowWindow 显示它。如果对话框是一个不定模式对话框，而且它不在最前面，就应使用窗口管理器的过程 ShowHide 显示它。

警告色彩表资源

在彩色显示器上，对话管理器自动地为您的警告栏和对话框添加色彩，使得它们和系统软件使用的窗口和警告信号的色彩匹配。当您创建警告资源时，您的应用程序的警告栏使用系统的系统设定色彩。通常您只需要为您的警告栏提供色彩。(但是，若要把一个图标的彩色版包括在警告栏中，您必须添加一个 'cicn' 类型的资源，它和项目列表资源中指定的黑白 'ICON' 资源有相同的资源标识符。)

如果您感到一定要使用非标准的色彩，可以使用对话管理器指定系统设定色彩以外的色彩。您的应用程序可以通过创建一个警告色彩表（ 'actb' ）资源为警告栏指定自己的色彩，这个资源必须和警告资源有同样的资源标识符（请参阅6-150 页的说明）。您不需要调用任何新的例程改变警告栏或对话框所使用的色彩。例如，当您调用Alert 函数时，对话管理器自动地试图装入一个和警告资源有同样资源标识符的警告色彩表资源。

但是，必须注意，您的警告栏中的非标准色彩可能在一开始使用户感到迷惑。还应注意，无论您作了什么改变，用户总是可以通过改变彩色控制板上的设定值改变警告栏的色彩。

警告:

因为彩色警告栏、彩色对话框、彩色项目和彩色图标在运行 System 7 以前的系统软件的电脑上是不可靠的，如果您希望保留和以前系统的兼容性，请不要创建这些色彩。

警告色彩表资源和窗口色彩表（即 'wctb' 类型的资源）的格式完全相同，后者的说明在本书的“窗口管理器”一章中给出。

项目色彩表资源

在彩色显示器上，对话管理器自动地在您的警告栏和对话框中绘制项目，使得它们和系统软件在它的警告栏和对话框中使用的色彩匹配。对话管理器在绘制您的对话框或警告栏的静态文字和可编辑文字项目中的文字时，也使用系统设定字体。

如果您感到一定要使用非标准的字体和色彩，可以使用对话管理器指定您自己的色彩、字体、字形和字体大小。

附注:

对话管理器仅在彩色显示器上显示您指定的字体、字形和字体大小。

您的应用程序可以通过创建一个项目色彩表（ 'ictb' ）资源指定这些内容，该资源必须和对话框或警告栏的项目列表资源有相同的资源标识符，然后，您应该为对话框提供一个对话项目色彩表资源，或为警告栏提供一个警告色彩表资源。您不需要调用任何新的例程改变对话框中使用的色彩、字体、字形或字体大小。例如，当您调用 GetNewDialog 函数时，对话管理器自动地试图装入一个和项目列表资源有同样资源标识符 的项目色彩表资源。

附注:

为了方便把您的应用程序在其它语系系统中本地化，您不应该改变字体。不要使用较小的字体，例如 9 点 Geneva 字体；某些语系 系统，例如 KanjiTalk 要求 12 点的字体。

还必须注意，您的对话框和警告栏中的项目使用非标准色彩，可能在一开始使用户感到迷惑。

警告

因为彩色警告栏、彩色对话框、彩色项目和彩色图标在运行 System 7 以前的系统软件的电脑上是不可靠的，如果您希望保留和以前系统的兼容性，请不要创建这些色彩。

如果您希望为一个警告栏或对话框提供项目色彩表资源，就必须创建一个警告色彩表资源或对话项目色彩表资源，即使项目色彩表资源只是说明静态文字和可编辑文字项目字形的改变，并不包含真正的彩色信息，您也必须这样做。

项目色彩表资源是一个'ictb'类型的资源。所有项目色彩表资源的资源标识符 都必须大于 128。

创建项目色彩表资源，没有 Rez 模板可用。当您编译一个项目色彩表资源时，应该遵循图 6-49 给出的格式。

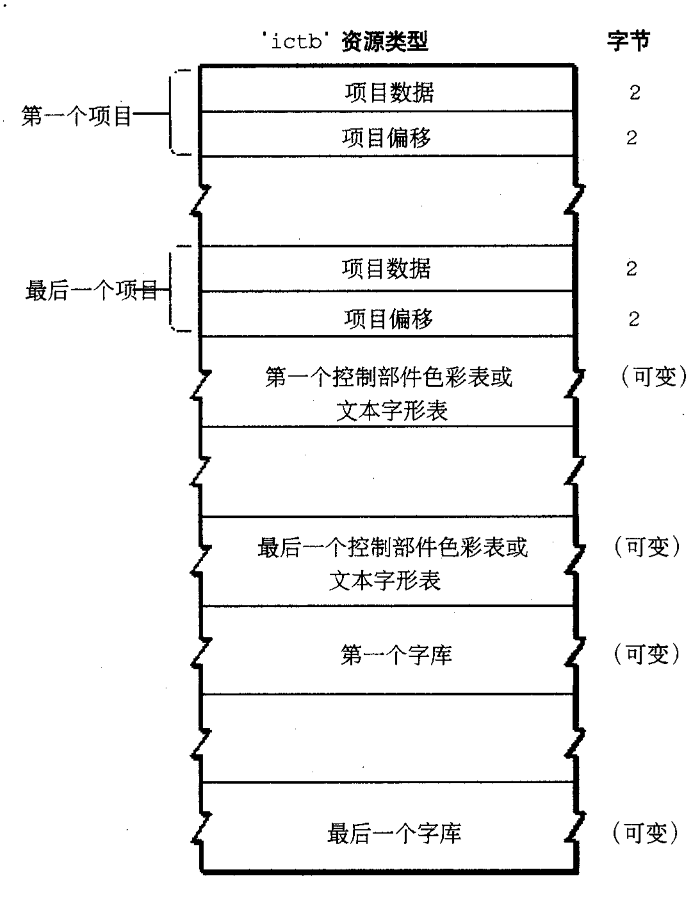


图 6-49 编译的项目色彩表资源的结构

您通过在一个 'ictb' 类型的资源中指定下列元素，为对话框或警告栏定义一个项目色彩表资源：

- 项目。它由可变数目的项目组成，它们相应于和该项目色彩表资源有同样资源标识符的项目列表资源中的项目

- 控制部件色彩表和文本字形表
  - ·控制部件色彩表定义了在控制部件中使用的色彩。几个控制部件可以共用一个控制部件色彩表。
  - ·文本字形表定义了可编辑文字项目或静态文本项目的字库、字形、字体大小和文字的色彩。几个可编辑文字项目和静态文本项目可以共用一个文字字形表。

- 您也可以选用一个字库列表。如果您使用任何文本字形表，通常会用一个文字字符串结束该项目色彩表资源，每个文字字符串指定了一个字库。尽管您也可以不用字体名称，而指定字体编号，但是指定字体名称更为可靠，因为在安装或删除这些字体时，系统软件可能会重新为它们编号。对于列在项目色彩表资源顶部的每个可编辑文字项目和静态文本项目，都应该在资源的底部指定一个字库

元素中包含的信息取决于它说明的项目类型：

- 项目数据。它包含了有关在该资源的其余部分是如何说明这个项目的信息
  - · 对于一个控制部件，这是它的控制部件色彩表的长度（以字节为单位）。
  - ·对于可编辑文字项目或静态文本项目，这个元素的位决定了使用文字体式表的哪一个元素，并按如下解释：

位	**意义**
0	改变字库
1	改变字体
2	改变字体大小
3	改变字体前台色彩
4	添加字体大小
13	改变字体字体背景色
14	改变字体模式
15	字体元素是对名称的一个偏移

- 项目偏移。从资源开始到说明该项目的控制部件色彩表或文本字形表的字节数

当项目数据和项目偏移都被设定为 0 时，就用系统设定的色彩、字体、字体大小和字体字形绘制控制部件或文字项目。即使对话框只有前几个项目有彩色字形的信息，对该方框中的所有项目都必须留有空间(未使用输入的项目数据和项目偏移元素设定为 0)。

控制部件的色彩用和控制管理器使用的‘cctb’资源完全一样的色彩表来说明。多个控制部件可以使用同样的色彩表。如果该资源把项目数据和项目偏移元素都设定为 0，该控制部件就使用系统的系统设定色彩。控制部件色彩表的格式请参阅图 6-50。

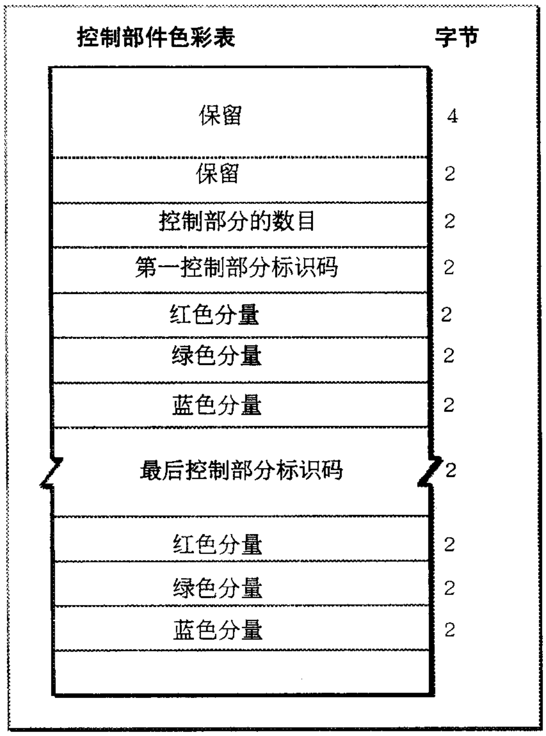


图 6-50　编译了的控制部件色彩表的结构

控制部件色彩表由下列元素组成：

- 保留。它的值应该总是设定为 0
- 保留。它的值也应该总是设定为 0
- 控制部分的数目。对于滚动杆以外的标准控制部件，它应该设定为 3，因为标准控制部件只使用 3 个部分：框架、控制部件主体和文字。对于滚动杆，它应该设定为 12。有关指定滚动杆色彩的信息，请参阅“控制管理器”一章中对于控制色彩表的说明。若要创建一个使用其它部分的控制部件，您必须创建一个自定义的 'CDEF' 资源，请参阅本书的“控制管理器”一章中的说明

控制部分标识码。这是标识第一控制部件的控制部分的值。下面的列表给出了代表滚动杆以外的标准控制部件的控制部分的值和常数。有关滚动杆的控制部分标识符的信息，请参阅本书的“控制管理器”一章。它们可以用任何次序在控制部件色彩表中列出。

常数	**值**	**控制部分**
cFrameColor	0	框架
cBodyColor	1	主体
cTextColor	2	文字（例如标题）

- 红色分量。它是一个整数，代表绘制该控制部分的色彩中红色分量的比例
- 绿色分量。它是一个整数，代表绘制该控制部分的色彩中绿色分量的比例
- 蓝色分量。它是一个整数，代表绘制该控制部分的色彩中蓝色分量的比例
- 下一个控制部分的标识码和红色、绿色及蓝色分量。您应该对要改变其色彩的控制部件的每一部分指定色彩分量。如果一个控制部分未在控制部件色彩表中列出，对话管理器就用系统设定色彩绘制它

图 6-51 给出了文本字形表的格式。

文本字形表	**字节**
字体	2
字形	2
字体大小	2
文字的红色分量	2
文字的绿色分量	2
文字的蓝色分量	2
背景的红色分量	2
背景的绿色分量	2
背景的蓝色分量	2
模式	2

图 6-51 编译了的文本字形表的结构

如图 6-51 所示，文本字形表的长度必须是 20 字节。多个可编辑文字项目和静态文字项目可以使用同一个文本字形表。若要以标准的字体、色彩、字体大小和字形显示文字，应把该项目的项目数据和项目偏移元素设定为 0。即使文字字形表的有些域不用，也必须为它们分配空间。

文本字形表由下列元素组成［有关字库、字形和点数大小的讨论请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书］：

- 字体。这是所使用的字库的名称。如果项目数据元素的第 15 位被设定为 1，就表示这个元素包含相对于资源结尾处的字体名称元素的偏移（以字节为单位）。如果项目数据的第 0 位被设定为 1，就表示这个元素包含一个字库的编号，如果项目数据的第 0 位被设定为 0，就表示这个元素被设定为 0，因而使用系统系统设定字体
- 字形。这是使用的字形。如果项目数据的第 1 位被设定为 1，这个元素就使用低位字节的数位说明使用在文字中的字形。如果低位字节的所有数位都被设定为 0，就使用正常字型。位的数字和它们代表的字形如下：

位的值	**字形**
0	黑体
1	斜体
2	下划线
3	轮廓线
4	阴影
5	压缩
6	扩展

- 字体大小。这是字体的点数大小。如果项目数据元素的第 2 位被设定为 1，这个元素就包含一个代表点数大小的值。如果项目数据元素的第 4 位被设定为 1，这个元素就包含添加至文字当前点数大小的一个值。如果项目数据元素的第 0 位被设定为 0，这个元素就被设定为 0，因而使用系统字体大小（12 点）
- 文字红色分量。如果项目数据元素的第 3 位被设定为 1，这个元素就包含一个整数，它代表绘制文字使用的彩色中红色分量的比例
- 文字绿色分量。如果项目数据元素的第 3 位被设定为 1，这个元素就包含一个整数，它代表绘制文字使用的彩色中绿色分量的比例
- 文字蓝色分量。如果项目数据元素的第 3 位被设定为 1，这个元素就包含一个整数，它代表绘制文字使用的彩色中蓝色分量的比例
- 背景红色分量。如果项目数据元素的第 13 位被设定为 1，这个元素就包含一个整数，它代表绘制文字后面的背景所使用的彩色中红色分量的比例

■ 背景绿色分量。如果项目数据元素的第 13 位被设定为 1，这个元素就包含一个整数，它代表绘制文字后面的背景所使用的彩色中绿色分量的比例

■ 背景蓝色分量。如果项目数据元素的第 13 位被设定为 1，这个元素就包含一个整数，它代表绘制文字后面的背景所使用的彩色中蓝色分量的比例

■ 模式。如果项目数据元素的第 14 位被设定为 1，这个元素就包含一个整数，它代表字符在位图像中放置的方式。下面列出了对话管理器解释的值和代表它们的常数。有关源转移模式的讨论请参阅《Inside Macintosh 绘图》*(Inside Macintosh : Imaging)* 一书

常数	**值**
scrOr	1
srcXor	2
srcBic	3

对话管理器小结

Pascal 小结

常数

常数

```
{checking for AppendDITL, ShortenDITL, CountDITL using Gestalt function}
gestaltDITLExtAttr = 'ditl';     {Gestalt selector for AppendDITL, etc.}
gestaltDITLExtPresent = 0;       {if this bit's set, then AppendDITL, }
                                 { ShortenDITL, & CountDITL are available}
{item types for GetDialogItem, SetDialogItem}
    ctrlItem      = 4;           {add this constant to the next four constants}
    btnCtrl       = 0;           {standard button control}
    chkCtrl       = 1;           {standard checkbox control}
    radCtrl       = 2;           {standard radio button}
    resCtrl       = 3;           {control defined in a control resource}
    helpItem      = 1;           {help balloons}
    statText      = 8;           {static text}
    editText      = 16;          {editable text}
    iconItem      = 32;          {icon}
    picItem       = 64;          {QuickDraw picture}
    userItem      = 0;           {application-defined item}
    itemDisable   = 128;         {add to any of the above to disable it}

{item numbers of OK and Cancel buttons in alert boxes}
    ok    = 1;          {first button is OK button}
    cancel    = 2;      {second button is Cancel button}

{resource IDs of alert box icons}
    stopIcon = 0;
    noteIcon = 1;
    cautionIcon = 2;

{constants used for theMethod parameter in AppendDITL}
    overlayDITL        = 0;      {overlay existing items}
    appendDITLRight    = 1;      {append at right}
    appendDITLBottom   = 2;      {append at bottom}
{constants for procID parameter of NewDialog, NewColorDialog}
    dBoxProc           = 1;      {modal dialog box}
    noGrowDocProc      = 4;      {modeless dialog box}
    movableDBoxProc    = 5;      {movable modal dialog box}
```

数据类型

```
TYPE DialogPtr        =    WindowPtr;
ResumeProcPtr         =    ProcPtr;
SoundProcPtr          =    ProcPtr;
ModalFilterProcPtr=        ProcPtr;
DialogPeek            =    ^DialogRecord;
DialogRecord          =
RECORD
    window:  WindowRecord;       {dialog window}
    items:       Handle;         {item list resource}
    textH:       TEHandle;       {current editable text item}
    editField:   Integer;        {editable text item number minus 1}
    editOpen:    Integer;        {used internally}
    aDefItem:    Integer;        {default button item number}
END;

DITLMethod            =    Integer;
```

对话管理器例程

初始化对话管理器

```
Initializing the Dialog Manager
PROCEDURE InitDialogs       (resumeProc: ResumeProcPtr);
PROCEDURE ErrorSound        (soundProc: SoundProcPtr);
PROCEDURE SetDialogFont     (fontNum: Integer); {also spelled SetDAFont}
```

创建警告信号

```
{some routines have 2 spellings--see Table 6-1 for the alternate spellings}
FUNCTION Alert          (alertID: Integer; filterProc:
                        ModalFilterProcPtr): Integer;
FUNCTION StopAlert      (alertID: Integer; filterProc:
                        ModalFilterProcPtr): Integer;
FUNCTION NoteAlert      (alertID: Integer; filterProc:
                        ModalFilterProcPtr): Integer;
FUNCTION CautionAlert   (alertID: Integer; filterProc:
                        ModalFilterProcPtr): Integer;
FUNCTION GetAlertStage : Integer;
PROCEDURE ResetAlertStage;
```

创建和处理对话框

```
{some routines have 2 spellings--see Table 6-1 for the alternate spellings}
FUNCTION GetNewDialog       (dialogID: Integer; dStorage: Ptr;
                            behind:  WindowPtr): DialogPtr;
FUNCTION NewColorDialog     (dStorage: Ptr; boundsRect: Rect; title: Str255;
                             visible: Boolean; procID: Integer; behind:
                             WindowPtr; goAwayFlag: Boolean; refCon:
                             LongInt; items: Handle): DialogPtr;
FUNCTION NewDialog          (dStorage:  Ptr; boundsRect: Rect; title: Str255;
                             visible: Boolean; procID: Integer; behind:
                             WindowPtr; goAwayFlag: Boolean; refCon:
                             LongInt; items: Handle):  DialogPtr;
PROCEDURE CloseDialog       (theDialog: DialogPtr);
PROCEDURE DisposeDialog     (theDialog: DialogPtr);
```

操纵警告栏和对话框中的项目

```
{some routines have 2 spellings--see Table 6-1 for the alternate spellings}
PROCEDURE GetDialogItem     (theDialog: DialogPtr; itemNo: Integer;
                            VAR itemType: Integer; VAR item: Handle;
                            VAR box: Rect);
PROCEDURE SetDialogItem     (theDialog: DialogPtr; itemNo: Integer; itemType:
                            Integer; item: Handle; box: Rect);
PROCEDURE HideDialogItem    (theDialog: DialogPtr; itemNo: Integer);
PROCEDURE ShowDialogItem    (theDialog: DialogPtr; itemNo: Integer);
FUNCTION FindDialogItem     (theDialog: DialogPtr; thePt: Point): Integer;
PROCEDURE AppendDITL        (theDialog: DialogPtr; theDITL: Handle;
                            theMethod: DITLMethod);
PROCEDURE ShortenDITL       (theDialog: DialogPtr; numberItems: Integer);
FUNCTION CountDITL          (theDialog: DialogPtr): Integer;
```

处理警告栏和对话框中的文字

```
{some routines have 2 spellings--see Table 6-1 for the alternate spellings}
PROCEDURE ParamText         (param0: Str255; param1: Str255;
                             param2: Str255; param3: Str255);
PROCEDURE GetDialogItemText     (item: Handle; VAR text: Str255);
PROCEDURE SetDialogItemText     (item: Handle; text: Str255);
PROCEDURE SelectDialogItemText
                            (theDialog: DialogPtr; itemNo: Integer; strtSel:
                            Integer; endSel: Integer);
PROCEDURE DialogCut         (theDialog: DialogPtr);
PROCEDURE DialogCopy        (theDialog: DialogPtr);
```

```
PROCEDURE DialogPaste       (theDialog: DialogPtr);
PROCEDURE DialogDelete      (theDialog: DialogPtr);
```

处理对话框中的事件

```
{some routines have 2 spellings--see Table 6-1 for the alternate spellings}
PROCEDURE ModalDialog    (filterProc: ModalFilterProcPtr; VAR itemHit: Integer);
FUNCTION IsDialogEvent   (theEvent: EventRecord): Boolean;
FUNCTION DialogSelect    (theEvent: EventRecord; VAR theDialog: DialogPtr; VAR
                         itemHit: Integer): Boolean;
PROCEDURE DrawDialog     (theDialog: DialogPtr);
PROCEDURE UpdateDialog   (theDialog: DialogPtr; updateRgn: RgnHandle);
```

应用程序定义的例程

```
PROCEDURE MyItem            (theWindow: WindowPtr; itemNo: Integer);
PROCEDURE MyAlertSound      (soundNo: Integer);
FUNCTION MyEventFilter      (theDialog: DialogPtr; VAR theEvent:
                            EventRecord; VARitemHit: Integer): Boolean;
```

C 小结

常数

```
enum {
/*checking for AppendDITL, ShortenDITL, CountDITL using Gestalt function*/
    #define gestaltDITLExtAttr 'ditl'/*Gestalt selector*/
    gestaltDITLExtPresent = 0    /*if this bit's set, then AppendDITL, */
                                 /* ShortenDITL, & CountDITL are available*/
};

enum {
/*item types for GetDItem, SetDItem*/
    ctrlItem      = 4,     /*add this constant to the next four constants*/
    btnCtrl       = 0,     /*standard button control*/
    chkCtrl       = 1,     /*standard checkbox control*/
    radCtrl       = 2,     /*standard radio button*/
    resCtrl       = 3,     /*control defined in a control resource*/
    statText      = 8,     /*static text*/
    editText      = 16,    /*editable text*/
    iconItem      = 32,    /*icon*/
```

```
    picItem       = 64,    /*QuickDraw picture*/
    userItem      = 0,     /*application-defined item*/
    helpItem      = 1,     /*help balloons*/
    itemDisable   = 128,   /*add to any of the above to disable it*/

/*item numbers of OK and Cancel buttons in alert boxes*/
    ok            = 1,     /*first button is OK button*/
    cancel        = 2,     /*second button is Cancel button*/

/*resource IDs of alert box icons*/
    stopIcon      = 0,
    noteIcon      = 1,
    cautionIcon   = 2
};

enum {
/*constants used for theMethod parameter in AppendDITL*/
    overlayDITL       = 0,    /*overlay existing items*/
    appendDITLRight   = 1,    /*append at right*/
    appendDITLBottom  = 2     /*append at bottom*/
};

enum {
/*constants for procID parameter of NewDialog, NewColorDialog*/
    dBoxProc          = 1,    /*modal dialog box*/
    noGrowDocProc     = 4,    /*modeless dialog box*/
    movableDBoxProc   = 5     /*movable modal dialog box*/
};
```

数据类型

```
typedef WindowPtr DialogPtr;

typedef struct DialogRecord DialogRecord;
typedef struct DialogRecord *DialogPeek;

struct DialogRecord                          {
        WindowRecord window;      /*dialog window*/
        Handle       items;       /*item list resource*/
        TEHandle     textH;       /*current editable text item*/
        short        editField;   /*editable text item number minus 1*/
        short        editOpen;    /*used internally*/
        short        aDefItem;    /*default button item number*/
};
```

```
typedef pascal void (*ResumeProcPtr)(void);
typedef pascal void (*SoundProcPtr)(void);
typedef pascal Boolean (*ModalFilterProcPtr)(DialogPtr theDialog,
                        EventRecord *theEvent, short *itemHit);
typedef short DITLMethod;
```

对话管理器例程

初始化对话管理器

```
pascal void InitDialogs      (ResumeProcPtr resumeProc);
pascal void ErrorSound       (SoundProcPtr soundProc);
pascal void SetDialogFont    (short fontNum); /*also spelled SetDAFont*/
```

创建警告信号

```
/*some routines have 2 spellings--see Table 6-1 for the alternate spellings*/
pascal short Alert           (short alertID, ModalFilterProcPtr filterProc);
pascal short StopAlert       (short alertID, ModalFilterProcPtr filterProc);
pascal short NoteAlert       (short alertID, ModalFilterProcPtr filterProc);
pascal short CautionAlert    (short alertID, ModalFilterProcPtr filterProc);
#define GetAlertStage() (*   (short*) 0x0A9A);
pascal void ResetAlertStage  (void);
```

创建和处理对话框

```
/*some routines have 2 spellings--see Table 6-1 for the alternate spellings*/
pascal DialogPtr GetNewDialog
                        (short dialogID, void *dStorage,
                        WindowPtr behind);
pascal DialogPtr NewColorDialog
                        (void *dStorage, const Rect *boundsRect,ConstStr255Param
                        title, Boolean
                        visible,short procID, WindowPtr behind,
                        Boolean goAwayFlag, long refCon, Handle items);
pascal DialogPtr NewDialog
                        (void *dStorage, const Rect *boundsRect,ConstStr255Param
                        title, Boolean visible, short procID,
                        WindowPtr behind,
                        Boolean goAwayFlag, long refCon,
                        Handle items);
pascal void CloseDialog      (DialogPtr theDialog);
pascal void DisposeDialog    (DialogPtr theDialog);
```

操纵警告栏和对话框中的项目

```
/*some routines have 2 spellings--see Table 6-1 for the alternate spellings*/
pascal void GetDialogItem    (DialogPtr theDialog, short itemNo,
                             short *itemType, Handle *item, Rect *box);
pascal void SetDialogItem    (DialogPtr theDialog, short itemNo, short itemType,
                              Handle item, const Rect *box);
pascal void HideDialogItem   (DialogPtr theDialog, short itemNo);
pascal void ShowDialogItem   (DialogPtr theDialog, short itemNo);
pascal short FindDialogItem  (DialogPtr theDialog, Point thePt);
pascal void AppendDITL       (DialogPtr theDialog, Handle theDITL, DITLMethod
                              theMethod);
pascal void ShortenDITL      (DialogPtr theDialog, short numberItems);
pascal short CountDITL       (DialogPtr theDialog);
```

处理警告栏和对话框中的文字

```
/*some routines have 2 spellings--see Table 6-1 for the alternate spellings*/
pascal void ParamText   (ConstStr255Param param0,
                        ConstStr255Param param1,
                        ConstStr255Param param2,
                        ConstStr255Param param3);
pascal void GetDialogItemText
                        (Handle item, Str255 text);
pascal void SetDialogItemText
                        (Handle item, ConstStr255Param text);
pascal void SelectDialogItemText
                        (DialogPtr theDialog, short itemNo,
                        short strtSel, short endSel);
pascal void DialogCut   (DialogPtr theDialog);
pascal void DialogCopy  (DialogPtr theDialog);
pascal void DialogPaste (DialogPtr theDialog);
pascal void DialogDelete(DialogPtr theDialog);
```

处理对话框中的事件

```
/*some routines have 2 spellings--see Table 6-1 for the alternate spellings*/
pascal void ModalDialog      (ModalFilterProcPtr filterProc, short *itemHit);
pascal Boolean IsDialogEvent (const EventRecord *theEvent);
pascal Boolean DialogSelect  (const EventRecord *theEvent,
                             DialogPtr *theDialog, short *itemHit);
pascal void DrawDialog       (DialogPtr theDialog);
pascal void UpdateDialog     (DialogPtr theDialog, RgnHandle updateRgn);
```

应用程序定义的例程

```
pascal void MyItem            (WindowPtr theWindow, short itemNo);
pascal void MyAlertSound      (short soundNo);
pascal Boolean MyEventFilter (DialogPtr theDialog, *EventRecord theEvent,
                              *short itemHit);
```

汇编语言小结

数据结构

DialogRecord 数据结构

0	dWindow	156 字节	警告栏或对话框的窗口记录
156	items	长型	警告栏或对话框的项目列表资源的间接指针
160	teHandle	长型	当前可编辑文字项目的间接指针
164	editField	字	当前可编辑文字项目
166	editOpen	字	内部使用
168	aDefItem	字	系统设定按钮的项目编号

全局变量

DAStrings	用 ParamText 过程指定的文字字符串的间接指针
DABeeper	当前语音过程的地址
DlgFont	对话框和警告栏中文字的字体编号
Acount	上一个警告信号的警告级编号（0 到 3）
ANumber	上一个警告信号的资源标识符
ResumeProc	继续执行过程的地址（不应在 System 7 中使用）

第七章 Finder 界面

Finder 界面介绍 7-3
关于 Finder 界面 7-6
使用 Finder 界面 7-6
给您的应用程序署名，给您的文稿设定创建者和文件类型 7-8
为 Finder 创建图标 7-11
创建自定的文稿图标 7-17
创建文件参考资源 7-18
创建束资源 7-20
Finder 打开您的应用程序的方式和时间 7-25
Finder 找不到您的应用程序时显示消息 7-27
提供版本资源 7-31
使用目录文件中的 Finder 信息 7-32
支持固定样板文稿 7-34
分配字体、语音及其它可移动资源 7-36
为非文稿图标提供旁述 7-38
使用替身 7-39
使用系统文件夹及和它有关的目录 7-41
桌面数据库 7-45
Finder 界面参考资料 7-46
数据结构 7-46
文件信息记录 7-47
扩展的文件信息记录 7-49
目录信息记录 7-50
扩展的目录信息记录 7-50
例程 7-51
分辨替身文件 7-51
7-53

资源 7-56
署名资源 7-57
图标列表资源 7-57
小图标列表资源 7-58
大 4 位彩色图标记录 7-59
小 4 位彩色图标记录 7-60
大 8 位彩色图标记录 7-61
小 8 位彩色图标记录 7-62
图标资源 7-63
彩色图标资源 7-64
文件参考资源 7-64
束资源 7-65
未找到应用程序名称字符串 7-68
未找到应用程序消息字符串 7-68
版本资源 7-69
Finder 界面小结 7-71
Pascal 小结 7-71
常数 7-71
数据类型 7-73
例程 7-74
C 小结 7-74
常数 7-74
数据类型 7-76
例程 7-77
汇编语言小结 7-77
数据结构 7-77
结果代码 7-78

Finder 是一个应用程序，它和系统软件一起跟踪文件并管理用户的桌面显示。本章介绍您的应用程序为了和 Finder 对话，应该使用的编程界面。

为了理解本章，您应该熟悉资源管理器。有关资源的一般信息，请参阅本书的“Macintosh 工具箱介绍”一章；有关资源管理器及其例程的详细信息，请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 一书中的“资源管理器”一章。事实上，为 Macintosh 电脑编写的任何软件都必须使用本章所介绍的和 Finder 有关的资源。

您应该通过阅读本章掌握以下内容：

- 设置 Finder 需要的资源，以便显示并启动您的应用程序
- 设置 Finder 使用的资源，以便显示和您的应用程序有关的的其它文件的信息
- 检查或改变储存在一个卷的目录文件中的有关 Finder 的信息
- 支持固定样板文稿
- 使用在系统文件夹中组织的目录

本章不解释如何使用苹果事件和 Finder 对话。当用户从 Finder 打开或打印一个文件时，Finder 把信息发送给您的应用程序，使它能够打开或打印该文件。在 System 7 中，支持高级事件的应用程序通过要求的苹果事件收到这个信息。

当 Finder 发送给您的应用程序这些要求的苹果事件时，它应该如何响应，请参阅《Inside Macintosh 应用程序间通信》*(Inside Macintosh: Interapplication Communication)* 一书，这些苹果事件包括：打开应用程序打开文稿、打印文稿和退出应用程序这四个基本的 Apple 事件。此外，您的应用程序可以使用另一组苹果事件请求 Finder 的服务，这组事件被称为 Finder 事件。例如，您的应用程序可以请求 Finder 执行代表您打开另一个应用程序这样一类操作。有关的详情，请参阅《Inside Macintosh 应用程序间通信》*(Inside Macintosh: Interapplication Communication)* 一书。

Finder 界面介绍

Finder 是管理用户桌面界面的应用程序。桌面环境是在 Macintosh 电脑上显示的工作环境——即，屏幕的灰色背景面积。

在桌面环境中，Finder 显示代表您的应用程序和它创建的文稿的图标，Finder 还跟踪用户的活动。图标是 Finder 用来表示一些用户可以操纵的对象——例如文件、文件夹、或废纸篓——的图像。例如，图 7-1 给出了 Finder 为几个示例应用程序（名为 SurfWriter 3.0，SurfPainter，和 SurfDB）和文字文稿（名为 Some Memo）所显示的图标，这个文字文稿是用户使用应用程序 SurfWriter 创建的。这些图标在一个窗口中显示，Finder 用这个窗口显示标签名为 Essentials 的磁盘图标的内容。

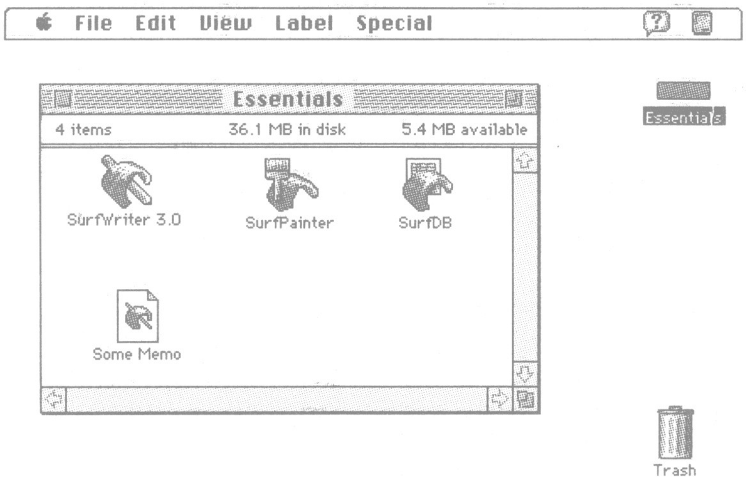


图 7-1 桌面环境的一个窗口中的应用程序和文稿图标

为了使用户能够区分您的产品,您应该为您自己的应用程序所创建的文件设计自己的图标。对于您的应用程序创建或使用的每一个文件类型,应该在单独的资源中分别定义大黑白图标、小黑白图标、大 4 位彩色图标、小 4 位彩色图标、大 8 位彩色图标和小 8 位彩色图标。然后您可以使用另一个资源，名为束资源，把这些图标指派给一种特定类型的所有文件。例如，图 7-1 中表示 Some Memo 的文稿图标是应用程序 SurfWriter 指派给它创建的所有文本文件的图标。用户用鼠标单击 SomeMemo 的图标，请求 Finder 打开应用程序 SurfWriter，SurfWriter 随后在窗口中打开文稿 Some Memo 作为响应。

固定样板文稿是用户作为其它文稿的模板创建的文件。出版数据是包含应用程序间共享的数据的特殊文件。询问文稿包含命令和数据，它的格式适用于数据库或其它数据源。如果您的应用程序支持这些文稿类型中的任何一种，您可以为 Finder 创建图标，来区分用户用您的应用程序所创建的固定样板文稿、出版数据和询问文稿。例如，本书封面的版面4显示了自定的固定样板文稿和出版数据图标，它们表示用户使用应用程序 SurfWriter 所创建的文稿。[有关出版数据请参阅《Inside Macintosh 应用程序间通信》*(Inside Macintosh : Interapplication Communication)*，询问文稿请参阅《Inside Macintosh 通信》*(Inside Macintosh: Communications)* 一书。]

您可能也希望您的应用程序为桌面环境中的文稿创建自定图标。或者您不产生一个应用程序，而是产生并分配其它应用程序使用的信息文稿(例如数据库文件、固定样板文稿、查询文稿、剪辑艺术库或字典)，您还可以为 Finder 提供区分您的文稿的自定图标。

Macintosh用户可以获取旁述形式的联机帮助。您可以自定您的应用程序图标显示的旁述。例如，图 7-2 给出了为 SurfWriter 应用程序图标自定的旁述。

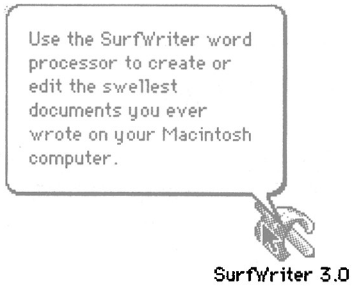


图 7-2 为一个应用程序图标自定的旁述

Finder 总是在适当的时候启动您的应用程序，并使用苹果事件通知您的应用程序应打开或打印的文稿。Finder 依靠您通过资源提供的信息执行这些任务。当用户创建或安装了一个文件时，文件管理器［请参阅《Inside Macintosh 文件》*(Inside Macintosh : Files)* 一书］开始在卷的目录文件中储存一部分信息。(目录文件是一种专门的文件，它位于卷内，包含该卷中有关文件和文件夹的分级组织的信息。)

Finder 从目录文件中析取您在资源中提供的信息，为了快速存取您的资源信息，Finder 使用这些信息为 2 MB (兆字节) 以上的卷建立一个桌面数据库，为 2 MB 以下的卷建立一个桌面文件。(桌面数据库是一个由 Finder 保留的图标、文件类型、应用程序、版本数据和备注组成的数据库。桌面文件是一个资源文件，Finder 在其中为 2 MB 以下的卷储存这些信息。)

当用户试图打开一个文稿，而您的应用程序不存在时，您甚至可以指定标识应用程序的资源。例如，如果用户试图打开一个名为 Instructions 的文稿，而应用程序 SurfWriter 未安装在用户的电脑系统中，Finder 就可以显示如图 7-3 的一个警告栏：

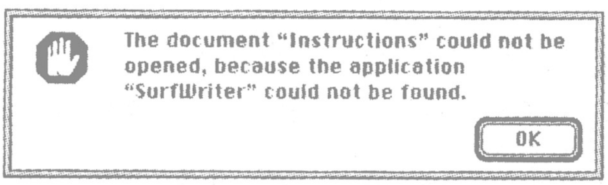


图 7-3 确认一个应用程序不存在的 Finder 消息

系统文件夹是一个目录，它包含着启动 Macintosh 电脑的软件。系统文件夹包括储存相关文件的一组文件夹。您的应用程序可以使用几个这样的文件夹储存文件。例如，您可能希望使用预置选择文件夹储存您的应用程序启动时需要的预置选择文件。

关于 Finder 界面

您可以使用 Finder 界面执行如下的任务：

- 创建资源（例如那些说明图标的资源），Finder 用这些资源析取并显示有关您的应用程序和它的文稿的信息(一般说来,所有的应用程序都应该为它们的文件提供这些资源)
- 确定及改变储存在目录文件中的 Finder 信息（一般说来，大多数应用程序都需要确定编目文件中的信息，许多应用程序可能会希望在目录文件中设定信息）
- 支持固定样板文稿，使得用户可以容易地对它们的文稿使用模板（一般说来，大多数创建文稿的应用程序应该支持固定样板文稿）
- 查找通常位于系统文件夹中的目录（一般说来，许多应用程序将希望存取这些目录）

使用 Finder 界面

Finder需要快速存取有关您的应用程序的一些关键信息,例如当显示您的应用程序和它的文稿时所使用的图标。您在应用程序文件的资源分支中提供这类信息中的大多数。

Finder 析取这个信息，并用它来构筑自己所需要的资源的数据库。Finder 在这个数据库中记录您的应用程序在磁盘上的位置,以便在用户打开您的一个文稿时,迅速地找到您的应用程序。

为了和 Finder 兼容，您的应用程序应该有如下的成分：

- 一个署名资源，它使 Finder 在用户双击您的应用程序创建的文稿时，能够标识并启动您的应用程序
- 一组说明图标的资源，这组图标代表了您的应用程序和它所创建的文稿在屏幕上的视觉形象
- 一组文件参考资源，它把图标和它们代表的文件类型链接起来，并允许用户通过把文稿图标拖拽至您的应用程序，打开您的应用程序
- 一个束资源，它把您的应用程序的署名资源、图标资源和文件参考资源组合在一起。
- 一个 'SIZE' 资源，它通知 Finder，当您的应用程序启动时，应该为它分配多少存储器，以及您的应用程序是否支持各种系统软件功能
- 在您的应用程序的文稿中的一个未找到应用程序名字字符串资源（如果用户试图在您的应 用程序不存在时，打开或打印您的应用程序创建的一个文稿，就显示它的名称），或是在您的应用程序的文稿中的一个未找到应用程序信息字符串资源（向用户解释为什么不能打开或打印仅供您的应用程序使用的文稿）

附注:

您应该为用户用您的应用程序打开的文稿提供一个未找到应用程序名称字符串资源为您的应用程序需要使用的但用户不应该打开的文稿(例如预置选择文件),提供一个未找到应用程序信息字符串资源。在一个文稿中只应提供两者中的一个,决不应该都提供。

您的应用程序还可以使用下列这些资源:

- 版本资源,使得用户能够较容易地找到您的应用程序的版本信息,以及(如果适用的话)您的应用程序的文件超集的版本信息
- 一个帮助资源,Finder 使用它为您的应用程序、控制板、系统功能扩展或桌面附件图标显示您自定的旁述消息

如果您采用一个为其它应用程序使用的文稿的格式来出售或分配数据,那么就可以通过提供以下各项帮助用户:

- 一个适当的文件类型,它允许用户通过把您的文稿的图标拖拽至一个应用程序,或在应用程序的文件菜单中选择"打开"命令,从 Finder 中打开该文稿
- 说明一个图标库的资源,对于用户,该图标库代表了您的文稿
- 一个未找到应用程序名字字符串资源或一个未找到应用程序说明信息字符串资源,使得 Finder 能够在用户试图打开或打印您的文稿时给予帮助
- 版本资源,使得用户能够容易地找到您的文稿的版本,如果您的文稿文件是一较大的文件集合,用户能够找到整个文件超集的版本

在每一卷中都有一个目录文件,它保留该卷中文件和目录间的关系。(一个卷是指格式化后包容文件的储存介质。)虽然目录文件主要由文件管理器使用,它也包含了 Finder 使用的信息。您也可以检查目录文件中的信息。特别是,您可能希望检查一个文件的文件类型或创建者,或者检查或设定文稿的一个 Finder 标志。当打开文稿时,您的应用程序应该检查一个Finder 标志,确定该文稿是否为一个固定样板文稿,如果是,您的应用程序应该把文稿的内容拷贝到一个新的文稿,并在一个未定义标题的窗口中打开这个新文稿。

您的应用程序可能希望使用位于系统文件夹中的文件夹。您最有可能存取的是预置项目、临时项目和废纸篓。例如,您可能希望检查预置项目中是否有用户配置文件,希望在临时项目中创建一个临时文件,或者当您的应用程序在试图储存文件时发现磁盘空间不够时,您希望检查并向用户报告废纸篓目录中的项目占用了多少空间。您可以使用 FindFolder 函数获得路径信息,凭借这个信息才能存取这些和系统有关的目录。

在System 7中，用户可以创建被称为替身的Finder对象，它们可以用来帮助组织用户的文件。通常当用户希望打开或打印文件时，您的应用程序并不需要关心它们是不是替身，因为在把它们送给您的应用程序前已经对替身进行了分辨。但是，如果您的应用程序在操纵文件中绕过Finder［或者标准文件软件包，请参阅《Inside Macintosh 文件》*(Inside Macintosh：Files)* 一书］，它就应该自己使用替身管理器函数 ResolveAliasFile 检查并分辨替身。

本章的其余部分详细说明了如何在您的应用程序中使用这些Finder功能。

给您的应用程序署名，给您的文稿设定创建者和文件类型

Finder通过您的应用程序的署名标识它，署名是一个独有的 4 字符序列。这个署名必须和其它应用程序的署名不冲突。为了确保唯一性，您必须在苹果电脑公司的Macintosh开发商技术支持部登记您的应用程序的署名。

附注：

由于您自己的资源类型通常只在您的应用程序或文稿中使用，所以不需要给它们登记。

您必须在自己的资源文件中包括一个特殊的资源，把您的应用程序的署名作为它的资源类型。按照惯例，署名资源的资源标识符 是 0。通常署名资源包含一个字符串，该字符串指定了您的应用程序的名称、版本号和发行日期。如果您不通过一个版本资源（请参阅 7-31 页的“提供版本资源”一节）提供专门的版本信息，Finder在用户选定您的应用程序，并在文件菜单中选择“获取信息”命令时，就显示署名资源中的字符串。

程序清单7-1 以Rez 输入格式给出了一个署名资源。(Rez 是一个资源编译程序，由苹果电脑公司的 Macintosh Programmer'S Workshop [MPW] 提供，可以从 APDA 获得。)

程序清单7-1 Rez输入格式的一个署名资源

```
type 'WAVE' as 'STR ';                  /*WAVE is the signature*/
resource 'WAVE' (0, purgeable) {        /*resource ID is 0*/
     "SurfWriter 3.0 © 1992"            /*default Get Info string*/
};
```

附注：

只有署名资源并不足以建立您的应用程序的署名。您还必须提供一个束资源，请参阅 7-20 页的“创建束资源”一节。

每当您的应用程序创建一个文稿时，应给该文稿指派一个创建者和一个文件类型。正如 7-32 页的“使用目录文件中的Finder信息”一节中所介绍的那样，您的应用程序通常把它的署名设定为文稿的创建者。当用户双击一个文稿，或者选定它后在Finder的文件菜单中选择打开或打印时，Finder读取该文件的创建者域，找出文稿的创建者。然后Finder搜索具有这个署名的应用程序，当它找到这个应用程序后，就打开它。

例如，如果文稿的创建者是您的应用程序的署名，Finder就调用进程管理器启动您的应用程序。然后，Finder向您的应用程序传送它需要的信息，以便打开或打印该文稿；自从System 7 推出后，Finder开始使用苹果事件向您的应用程序传送这个信息。《Inside Macintosh 应用程序间通信》*(Inside Macintosh : Interapplication Communication)* 一书中说明了您的应用程序应该如何处理要求的苹果事件，以便打开或打印文件。

在 7-32 页的“使用目录文件中的Finder信息”一节中介绍到，您的应用程序在创建文稿时，通常为它指派一个文件类型。文件类型可以是专门为您的应用程序定义的类型，也可以是一般现有的类型，现列出如下:

文件类型	**说明**
'APPL'	可打开的应用程序
'DFIL'	储存桌面附件的文件
'DRVR'	驱动程序
'FFIL'	储存字体的文件
'INIT'	系统功能扩展
'PICT'	QuickDraw 图片
'PRER'	打印机驱动程序
'RDEV'	选择程序功能扩展
'TEXT'	ASCII 字符流
'adev'	网络功能扩展（例如 EtherTalk 2.0）
'appe'	仅限后台使用的应用程序（没有用户界面）
'cdev'	控制部件面板
'edtp'	共享面向图形数据的出版数据
'edts'	共享面向语音数据的出版数据
'edtt'	共享面向文字数据的出版数据
'ffil'	字体
'ifil'	语系系统资源集合
'kfil'	键盘布局
'pref'	预置选择文件
'qery'	存取数据库的询问文稿
'scri'	语系系统的系统功能扩展
'sfil'	语音

文件类型	**说明**
'tfil'	TrueType 字体
'ttro'	TeachText 只读文件
'zsys'	一个系统文件（例如系统文件本身）

附注:

完全由小写字母和非字母字符组成的署名和文件类型仅供苹果电脑公司使用。您创建的署名和文件类型至少必须包含一个大写字母。因为系统软件不会向用户显示署名和文件类型，署名和文件类型可以包含字符组合，它可以是除了您以外谁也不理解的组合。

文件类型和署名一样，也必须在苹果电脑公司登记。您的应用程序必须有一个文件类型 'APPL' 。您的应用程序文件的创建者域应该包含它自己的署名。大多数编程环境都提供了一个简单的工具设定您的应用程序文件的创建者域。

您的应用程序可以创建任意类型的文件，它可以指定任何应用程序作为创建者。例如，您应该编写一个实用应用程序，它通过打开一个文本文件并为它增补另一个文本文件，来创建一个新的文稿。应用程序将给新文稿和第一个原始文字文件同样的创建者，这样，当用户打开或打印新的文稿时，Finder 可以调用该应用程序。

对于仅由文本组成的文件，应指派标准的文件类型 'TEXT'，它是一个字符流，在每段的结尾有回车字符。大多数字处理程序允许用户创建只有文本的文件。一个 'TEXT' 类型的文稿可以用接受这种文件类型的任何应用程序打开或打印。您的应用程序仍然可以指派自己的署名作为文件的创建者，使得Finder可以在适当时调用它打开或打印该文件。

用户也可以通过选定一个文稿的图标，并把它拖拽至您的应用程序的图标，打开该文稿，只要这个文稿是由您的应用程序所创建的，或者它的文件类型是您的应用程序所支持的。因为文稿的文件类型储存在目录文件中，而Finder把您的应用程序所支持的文件类型的列表储存在桌面数据库中，所以Finder可以决定是否打开您的应用程序。如果您的应用程序支持文稿的文件类型，Finder打开您的应用程序，并向它传送该文稿的名称。（本章的以后各节将对这些问题进行详细的讨论。）

例如，如果您的应用程序是一个页面布局程序，它可能创建自己的文件类型，同时也支持 'TEXT' 和 'PICT' 文件类型的文稿。用户就可以通过把一个属于这些类型的文稿的图标拖拽至您的应用程序的图标来打开它。

您的应用程序也要依靠文件类型来确定它在运行时用户可以打开什么文件。当您的应用程序调用标准文件软件包打开文件时，它应该提供一个可以打开的文件列表，或者对于文件类型的一个过滤函数。然后，打开文件对话框就只显示指定类型的文件。[有关细节请参阅《Inside Macintosh 文本》(*Inside Macintosh: Files*) 一书。]

为 Finder 创建图标

Finder 用图标表示您的文件。为了使用户能够区分您的产品，您应该为与您的应用程序有关的所有文件设计自己的图标，它们包括：

- 您的应用程序自身的文件图标
- 您的应用程序创建的标准文稿图标
- 用户从您的应用程序的文稿所创建的固定样板文稿
- 用户从您的应用程序的文稿创建的数据共享的编辑数据
- 其它专门文件，例如只读文件，图形和询问文稿，它们是由您的 Macintosh 应用程序创建的，或是您提供为其它 Macintosh 应用程序使用的

对于大多数有效的显示，您应该为自己的每一个文件创建一个图标库。图标库是代表Finder所显示的一个对象的一组图标，这个对象可以是一个应用程序或一个文稿。一个完整的图标库由大（32 × 32 像素）图标和小（16X16 像素）图标组成，每一个都有反白图标，并有 3 种不同的彩色：黑白、每像素 4 位彩色数据、每像素 8 位彩色数据。例如，下面为一个文件的图标库：

- 大（32 × 32 像素）黑白图标和反白图标，您在一个图标列表（ 'ICN#' ）资源中定义它们
- 小（16 × 16 像素）黑白图标和反白图标，您在一个图标列表（ 'ics#' ）资源中定义它们
- 每像素 4 位彩色数据的大（32 × 32 像素）彩色图标，您在一个大 4 位彩色图标（ 'icl4' ）资源中定义它
- 每像素 4 位彩色数据的小（16 × 16 像素）彩色图标，您在一个小 4 位彩色图标（ 'ics4' ）资源中定义它
- 每像素 8 位彩色数据的大（32 × 32 像素）彩色图标，您在一个大 8 位彩色图标（ 'icl8' ）资源中定义它
- 每像素 8 位彩色数据的小（16 × 16 像素）彩色图标，您在一个小 8 位彩色图标（ 'ics8' ）资源中定义它

本书封面的版面 3 显示了示例应用程序 SurfWriter 如何使用这些资源为它的应用程序图标定义图标库。

和这些资源有点关系的是图标（ 'ICON' ）资源和彩色图标（ 'cicn' ）资源。您可以使用这两个资源中的任何一个说明您的应用程序某个元素中的 32 × 32 像素图标。但是，Finder 不会使用或显示您所创建的 'ICON' 或 'cicn' 类型的资源。在您的应用程序中使用这些资源显示图标的是您自己应用程序。一般说来，您在一个菜单或对话框中显示黑白图标，请参阅本书的“菜单管理器”和“对话管理器”两章。(例如，本书封面的版面 2 中的彩色警告栏为它左上角的彩色图标指定了 'cicn' 类型的资源。) 如果您提供一个彩色图标（ 'cicn' ）资源，其资源标识符和图标（'ICON'）资源一样，菜单管理器和对话管理器就为有彩色显示器的用户显示彩色图标，而不是黑白图标。

在为您的文件创建图标库之前，您应该先设计一个所有图标库都可以共享的图形元素，这个元素可以帮助用户很快地识别和您的产品有关的文件。例如，图 7-4 说明了一个公司如何把一个波状物的图像用于它的应用程序所有的图标中，这些图标代表了文字编辑应用程序 SurfWriter，图形应用程序 SurfPainter 和数据库应用程序 SurfDB。正如本书封面的版面 4 所示，波状元素也出现在代表用户使用这些应用程序所创建的文稿、固定样板文稿和出版数据中。

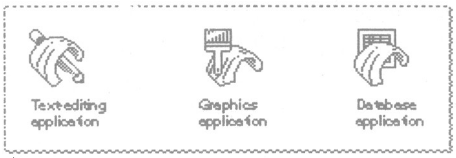


图 7-4　一个公司的产品系列的大黑白应用程序图标

如果您不设计自己的图标，Finder将使用它的一组应用程序和文稿的系统设定图标进行显示。图 7-5 显示了 Finder 的系统设定大黑白图标。



图 7-5 系统设定大黑白图标

附注:
用图 7-5 所示的系统设定图标显示的桌面附件是为旧版 Macintosh 系统软件设计的，这些系统软件不支持协同多任务环境。在 System 7 中，桌面附件和应用程序的外观和行为要相像得多。因为再创建桌面附件已经不太必要，所以，在您希望创建一个小的或简单的程序时，一般只需编写一个小应用程序，而不必创建一个桌面附件。

如果您不希望 Finder 为您的应用程序或文稿显示系统设定图标，您就必须至少为每个图标定义一个图标列表（'ICN#'）资源。

图标列表这个术语稍微有点用词不当，因为您只能在图标列表资源中定义两个图像，一个 32 × 32 像素的黑白图标和它的反白图标。正如本节后面所示，若要为一个文件定义彩色图和 16 × 16 像素的图标，您需要创建附加的资源。（如果不为您的图标定义彩色版，Finder 就在所有的显示中都使用您的图标列表资源中定义的黑白图标，如果您不定义 16 × 16 像素的图标，Finder 将在需要时通过计算把 32 × 32 像素的图标减小一半。）

一个图标列表资源定义了一个图标。它包含两个图标说明：显示用的真实图标和显示图标被选中时的反白图标。Finder 使用这个反白图标把图标的轮廓线隐藏到桌面的任何背景色或图案中。然后，Finder 把图标绘入这个外形。因此，反白图标必须和图标有完全一样的形状。反白图标还定义了用户需要的能够通过鼠标单击选定图标的面积。因而在反白图标中最好是全抹黑的，否则用户选定图标时会有麻烦。

图 7-6 给出了一个应用程序的黑白图标和它的反白图标。在这个示例图标和它的反白图标中，波状物下面笔周围的面积出现了问题：在图标中间的两个小区域就像反白图标中的未抹黑的空隙一样，在那里用户不能用光标选定这个图标。



图 7-6　一个应用程序的黑白图标和它的掩码

一个图标列表资源被定义为两个 String [128] 类型的项目的阵列，第一个阵列中的每个数位代表了 32 × 32 像素图标中的一个像素，第二个阵列中的每个数位代表了 32 × 32 像素反白图标中的一个像素。通常您使用一个高级工具，例如 ResEdit 图标编辑程序（可以通过 APDA 获得），创建您的图标列表资源。图 7-7 显示了如何用 ResEdit 图标编辑程序创建图 7-6 中的图标的图标列表资源。当您对图标的外观感到满意后，就可以使用 DeRez 反编译程序把您的图标列表资源转换成 Rez 输入。

程序清单 7-2 是图标列表资源的 Rez 输入的一部分，它说明了图 7-7 中的应用程序图标。程序清单 7-2 还给出了另一部分图标列表资源，它用于代表该应用程序创建的文稿的图标。这个列表和本章以后的列表都使用 Rez 输入格式，以便帮助您理解资源的格式，以及如何将它们合在一起工作。

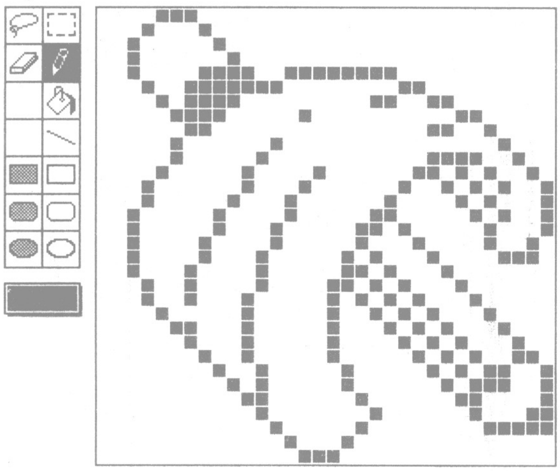


图 7-7 图标的 ResEdit 模式

程序清单 7-2 一个图标列表资源的 Rez 输入格式

```
data 'ICN#' (128, purgeable) {        /*application icon & mask*/
                  /*array: 2 elements*/
                  /*[1]: the application icon*/
    $"0E 00 00 00"/*1st line of icon: 4 bytes (32 bits)*/
    .             /*32 lines total in icon*/
    .
    .
    ,             /*[2]: the mask*/
    $"0E 00 00 00 /*1st line of mask: 4 bytes (32 bits)*/
    .             /*32 lines total in mask*/
    .
    .
```

```
};
data 'ICN#' (129, purgeable) {   /*text document icon and mask*/
                    /*icon data goes here*/
};
data 'ICN#' (130, purgeable) {  /*stationery pad icon & mask*/
                    /*icon data goes here*/
};
data 'ICN#' (131, purgeable) {  /*edition icon & mask*/
                    /*icon data goes here*/
};
```

您也可以在一个小图标列表资源（即，资源类型为 'ics#' 的资源）中定义您的小图标（16 × 16像素）。当用户在显示菜单中选择“按照小图标”显示文件夹中的内容时，Finder 会在窗口中显示应用程序或文稿的小图标。在黑白显示器上，用户打开您的应用程序后，小图标还会出现在应用程序菜单中，如果用户把您的应用程序或一个替身放置在苹果菜单命令项目文件夹的苹果菜单中，小图标也会在苹果菜单中出现。(有关替身文件和苹果菜单命令项目文件夹请分别参阅 7-39 页的“使用替身”和 7-41 页的“使用系统文件夹及和它有关的目录”两节。)

您还应该通过使用几种资源类型定义大小图标的彩色版本。一个图标的不同版本有同样的资源标识符，都和定义大黑白图标的图标列表资源的资源标识符一样。例如，如果应用程序图标的图标列表资源的资源标识符是 128，那么，它的小图标列表资源的资源标识符也应该是 128，该图标的下列资源的资源标识符都应该是 128：大 4 位彩色图标资源、小 4 位彩色图标资源、大 8 位彩色图标资源和小 8 位彩色图标资源。

不要为定义彩色图标的资源定义反白图标。大 4 位彩色图标资源和大 8 位彩色图标资源使用在它们各自的伴随图标列表资源中定义的黑白图符的反白图标，小 4 位彩色图标资源和小 8 位彩色图标资源使用在它们各自的伴随 'ics#' 资源中定义的黑白图标的反白图符。因此，您的彩色图标的轮廓线形状应该和您的 'ICN#' 和 'ics#' 资源中定义的完全匹配。

ResEdit2.1 包含一个图标库编辑程序,它能够帮助您容易地创建这些有关的资源。有关细节请参阅《ResEdit Reference》一书。

有关对您的图标最有效地使用彩色和形状的方法，请参阅《Macintosh 用户界面设计原则》*(Macintosh Human Interface Guidelines)* 一书。通常，您最好首先在图标列表资源和小图标列表资源中创建黑白图标，然后，使用定义彩色图标的资源为它们添加色彩。不要在这些资源中改变图标的形状，否则在图标列表资源和小图标列表资源中定义的反白图标会和这些形状不匹配。您应该在 36 种推荐的系统调色板图标彩色中选择您的颜色。(如果您使用 ResEdit 2.1，当您在彩色菜单中选择“苹果图标色彩”时，这些色彩就出现在一个调色板上。)请注意，您不能为这些资源指定自己的色彩表。

有关彩色调色板更详细的信息，请参阅《Inside Macintosh 绘图》*(Inside Macintosh : Imaging)* 一书。虽然当系统需要定义彩色环境时，调色板管理器允许您定义一个调色板供它使用，但是您的图标还是应该使用系统调色板的彩色。当您的应用程序未在运行时，用户仍可能经常使用Finder，当您的应用程序运行时，用户还可能切换到另一个应用程序。使用系统调色板的好处是，无论谁是激活的应用程序，您的图标在Finder中显得更和谐。而且，因为用户能够改变桌面彩色和图案，如果您的图标使用系统调色板，用户对于他们的工作环境就有更多的控制能力。用户总是可以改变您的彩色定义，他们只需选定一个图标并在标签菜单中选定一种色彩，然后Finder就把选定的色彩和选定的图标的色彩调和，得到新的色彩。若要还原原来的色彩，用户必须在标签菜单中选择“无”。

如果您的应用程序创建了文稿，它应该至少定义两个附加的图标库 一个代表您的应用程序创建的文稿，另一个在用户从该应用程序的一个文稿创建固定样板文稿时显示。(有关固定样板文稿请参阅 7-34 页的“支持固定样板文稿”一节。)

如果您的应用程序有多种文稿，您可以为不同的文稿提供不同的图标，这样能够使用户感到方便。例如，TeachText对于它的只读文稿和图形文稿就使用不同的图标库。

如果您的应用程序通过出版数据管理器支持数据共享，它还应该为出版数据定义一个图标库。出版数据管理器 [请参阅《Inside Macintosh 应用程序间通信》*(Inside Macintosh : Interapplication Communication)* 一书] 允许用户在不同的文稿和应用程序中共享并自动更新数据。例如，一个用户可能需要从一个电子表格中获取销售数据及总额，然后把这个信息包括在某月销售小结的文字处理文稿中。如果电子表格和字处理应用程序都支持出版数据管理器，用户就可以简单地通过下列步骤完成这一工作。首先，用户在电子表格中选定数据并创建一个版本。然后电子表格的应用程序在一个单独的文件中写该数据的一个拷贝，该文件被称为出版数据。这个出版数据用一个图标代表，7-12 页的图 7-5 中的出版数据图标就是它的系统设定图标。如果用户打开一个文字处理文稿并创建一个电子表格文稿出版数据的订户，字处理应用程序就可以从电子表格文稿出版数据中把需要的销售数据和总额结合到文字处理文稿中。

如果您的应用程序设计成自己创建出版数据，应考虑创建一个图标，它唯一地标示您的出版数据，并把它们和您的应用程序的文稿联系起来。包含出版数据的文件的类型应该是 'edtt' (面向文本的数据)、'edtp' (面向图形的数据)、或 'edts' (面向语音的数据)；当然，创建者应该是您的应用程序的署名。

如果您的Macintosh应用程序是一个数据库程序或是作为数据源(一个电子表格程序常起这样的作用)，您可能希望创建查询文稿，使得其它Macintosh 应用程序可以通过数据存取管理器存取这些数据。这时，您的应用程序也应该为它的查询文稿定义一个图标库。[有关在这种情况下共享数据的信息请参阅《Inside Macintosh 通信》*(Inside Macintosh : Communications)* 一书。]

本书封面的版面 4 显示了应用程序SurfWriter创建的各种文稿的大彩色图标 文本文稿、固定样板文稿和出版数据。

只定义图标资源对于显示您的图标还是不够的。您还必须执行下面两组过程之一：

- 如果您是应用程序开发商，就必须为您的应用程序定义文件参考资源和一个束资源。请参阅 7-18 页的“创建文件参考资源”和 7-20 页的“创建束资源”两节
- 如果您是一个信息提供商或数据库开发商，也就是说您提供文稿供其它应用程序使用，那么，在运行 System 7 的 Macintosh 电脑上，您就不需要创建文件参考资源或束资源来提供文稿图标。您可以使用下一节说明的方法为您的文稿创建自定的图标

创建自定的文稿图标

您可以为您的文稿创建自定图标。用户也可以创建自定图标。当一个资源标识符为-16455 的图标列表资源被储存在一个文件的资源分支中时，Finder 使用在该资源中定义的大小黑白图标、大小 4 位彩色图标和大小 8 位彩色图标作为定制图标，Finder 不使用系统设定图标，也不使用在文件的束资源中列出的任何图标。

附注：

虽然一个应用程序可以通过在一个束资源中把它的文稿的图标和文稿的文件类型联系起来，为它所有文稿指派图标（请参阅 7-18 页的“创建文件参考资源”和 7-20 页的“创建束资源”两节），一个自定图标只能代表一个特定的文件，该文件在它的资源分支中的图标列表资源的资源标识符是 -16455。

System 7的用户能够自定单独的图标。通过选定一个文件并在文件菜单中选择“获取信息”，用户可以看到该文件的信息窗口。然后用户可以选定在信息窗口左上角显示的图标，并使用编辑菜单中的“贴”命令用剪贴板中的图片代替它。Finder 根据用户的自定图标创建一个图标库，为该图标库中的每个资源指派资源标识符为 -16455，把这些资源储存在图标所代表的文件的资源分支中，并在文件的 Finder 标志域设定 hasCus-tomIcon。（有关 Finder 标志的详细信息，请参阅 7-47 页的“文件信息记录”一节。）

您的应用程序可以使用同样的方法为它创建的文稿提供自定图标。例如，一个绘图应用程序可能创建它的文稿中所包含的插图的缩影，并用它们作为文稿图标。

如果您是数据库开发商，创建并分配支持数据存取管理器的查询文稿，您也可以使用这个方法创建标示您的数据库查询文稿的图标。同样，即使您并不生产应用程序，而是生产并分布信息（例如数据库文件、固定样板文稿、剪辑艺术库和字典）供其它应用程序使用，您可能也希望为您的文稿提供独特的图标。

为了使Finder显示文稿的自定图标，您必须至少创建一个资源标识符 是 -16455 的图标列表资源，并把它储存在文稿的资源分支中。[若要在您的应用程序运行时创建这个资源，它应该调用 AddResource 过程，请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh : More Macintosh Toolbox)* 一书的"资源管理器"一章。] 您可以使用下面的常数代替标识符编号:

```
CONST    kCustomIconResource = -16455;{res ID for custom icon}
```

如果您只提供一个图标列表资源，Finder就在所有的屏幕上显示黑白图标，并在需要小图标时自动把它缩小。若要创建彩色图标及定义小图标，您应创建整个图标库，请参阅 7-11 页的"为 Finder 创建图标"一节。

在使用 kCustomIconResource 常数作为资源标识符创建了图标的资源后，您必须在文件的 Finder 标志域中设定 hasCustomIcon。为了防止用户改变这些图标，应在文件的Finder标志域中设∂®nameLocked。(大多数开发环境为设定这些标志位提供了工具。7-32 页的"使用目录文件中的Finder 信息"一节中说明了如何确定及设定这些 Finder 标志。)

创建文件参考资源

文件参考 ('FREF') 资源起两种作用。第一，它们把您定义的图标和您的应用程序使用的文件类型联系起来; 第二，它们允许用户通过把文稿图标拖拽至您的应用程序图标，在您的应用程序中打开该文稿。

应该为您的应用程序文件自身创建一个文件参考资源，并为您的应用程序可以开启的每一种文件类型创建单独的文件参考资源。程序清单 7-3 以 Rez 输入格式给出了下列文件的文件参考资源: SurfWriter 应用程序文件、文字文稿、固定样板文稿、和出版数据，以及 TeachText 只读文稿。

每个文件参考资源指定了下列项目:

- 一个文件类型
- 在束资源中指定的图标列表资源的局部资源标识符
- 一个空字符串

您只能为以下文件定义文件类型 您的应用程序创建的文件、您的应用程序所支持的其它应用程序创建的文件，或者现有的一般类型的文件，例如 'TEXT' 和 'PICT'。

在下一节"创建束资源"中将要说明，局部资源标识符把文件类型映射到一个图标列表资源，它在束资源中被指派了同样的局部资源标识符。例如，如果您希望两种文件类型共享同样的图标，您可以创建两个单独的文件参考资源，它们具有同样的局部资源标识符，这样束资源就把它们映射到同样的图标列表资源。(但是，两种文件类型共享同样的图标不是一种好的做法，因为用户使用 Finder 时，将无法区分不同的文件类型。)

程序清单7-3 Rez输入格式的文件参考资源

```
resource 'FREF' (208, purgeable) {        /*SurfWriter application*/
     'APPL',        /*type 'APPL'*/
     0,        /*maps to icon list resource w/ local ID 0 in bundle resource*/
     ""             /*leave empty string for name: not implemented*/
};
resource 'FREF' (209, purgeable) {        /*SurfWriter document*/
     'TEXT',        /*type 'TEXT'*/
     1,        /*maps to icon list resource w/ local ID 1 in bundle resource*/
     ""
};
resource 'FREF' (210, purgeable) {        /*SurfWriter stationery pad*/
     'sEXT',        /*type 'sEXT'*/
     2,        /*maps to icon list resource w/ local ID 2 in bundle resource*/
     ""
};
resource 'FREF' (211, purgeable) {        /*SurfWriter edition*/
     'edtt',             /*type 'edtt'*/
     3,        /*maps to icon list resource w/ local ID 3 in bundle resource*/
     ""
};
resource 'FREF' (212, purgeable) {        *TeachText read-only files*/
     'ttro', 4, ""              /*These documents have TeachText as their */
                   /* creator. Finder uses TeachText's icon list resource */
                             /* for these documents. Included here so users */
                             /* can drag these docs to SurfWriter's app icon*/
};
```

如果您为固定样板文稿提供您自己的图标(该固定样板文稿是用户从您的应用程序文稿中创建的),就应该为它创建一个文件参考资源。请按照下述方式为这个文件参考资源指派一个文件类型:使用创建该固定样板文稿所根据的文稿的文件类型,但是把该文稿的文件类型的第一个字母用小写 s 代替。您把它映射到束资源中的一个图标列表资源,这是和其它文件参考资源一样的。(由于这个惯例,您的文稿文件类型必须由它的最后 3 个字母唯一决定。)

例如,在程序清单 7-3 中,文件参考资源中指派的 'SEXT' 文件类型被用于从 'TEXT' 文件类型的文稿中创建的固定样板文稿。在这种情况下,即文件的 Finder 标志域设定为 isStationery,Finder 显示应用程序 SurfWriter 的束 ('BNDL') 资源,确定映射到类型为 'SEXT' 的文稿的图标。然后,Finder 用本书封面的版面4 中显示的固定样板文稿图标显示这类文稿。

当用户把一个文稿的图标拖拽至您的应用程序图标时，Finder就检查它保留的一个您的文件参考资源列表。如果文稿的文件类型出现在这个列表中，Finder就开启您的应用程序并要求它打开该文稿。

如果您的应用程序支持某些文件类型，但不为它们提供图标，您仍然可以对它们定义文件参考资源，这样，用户可以把这些文稿的图标拖拽到您的应用程序图标中，打开您的应用程序。例如，7-19 页的程序清单 7-3 中创建了具有资源标识符 212 的文件参考资源，这样，当用户把 TeachText 的只读文稿的图标拖拽到应用程序 SurfWr-iter 的图标中时，Finder 就打开应用程序 SurfWriter。因为这些文稿的创建程序为 TeachText，Finder 就显示应用程序 TeachText 在自己的束资源中为它们定义的图标。

通过支持打开文稿事件，您还可以在文件参考资源中指定磁盘、文件夹和一对通配符文件类型，使得用户可以通过把他们的图标拖拽至您的应用程序图标，打开您的应用程序。打开文稿事件是 4 种基本的苹果事件之一，《Inside Macintosh 应用程序间通信》(*Inside Macintosh : Interapplication Communication*) 一书对此作了解释。在 Finder使用进程管理器打开一个支持高级事件的应用程序后，它就向您的应用程序发送一个打开文稿事件，该事件包括应用程序应该打开的对象的一个替身记录列表。

因为替身记录能够指定卷、目录以及文件，所以打开文稿事件给了您一个机会，处理用户把磁盘和文件夹图标拖拽至您的应用程序的情况。(7-39 页的“使用替身”一节将对替身记录进行说明。) 您可以创建一个文件参考资源并指定 'disk ' 作为文件类型，使用户能够把硬盘和软盘图标拖拽至您的应用程序图标中。您也可以创建一个文件参考资源并指定 'fold' 作为文件类型，使用户能够把文件夹图标拖拽至您的应用程序图标中。

您可以创建一个文件参考资源，它指定‘****’作为文件类型，这样用户就能够把所有的文件类型——包括应用程序、系统功能扩展、文稿等等，都拖拽制您的应用程序图标，但是不包括磁盘和文件夹。如果您创建 3 个文件参考资源，把 'disk'、'fold' 和 '****' 指定为它们的文件类型，而且您的应用程序支持打开文稿事件，用户就可以通过拖拽任何图标至您的应用程序图标，打开您的应用程序。您的应用程序应该按照用户的需要打开磁盘、文件夹、或所有可能的文件类型。

创建束资源

束 ('BNDL') 资源为您的应用程序把 Finder 使用的所有资源联系起来；特别是它把您的应用程序和它的文稿与它们的图标联系起来。束资源包含：

- 应用程序的署名
- 它的署名资源的资源标识符编号 (它应该总是为 0)
- 把局部资源标识符指派给对应用程序定义的所有图标列表资源的资源标识符；局部资源标识符必须和在相应的文件参考资源中指派的一样

为了保持兼容，把局部标识符 指派给文件参考资源标识符。(为了保持一致，它们可以是在文件参考资源中指派的同样的局部标识符，但是并非必须如此，它们只需对每个文件参考资源唯一即可。)

当Finder首次在用户的桌面环境中显示您的应用程序时，它检查目录文件（请参阅7-32 页的“使用目录文件中的Finder信息”一节)，核对您的应用程序是否有一个束资源。如果没有，Finder就显示 7-12 页的图 7-5 中的系统设定图标。如果您的应用程序有一个束资源，Finder安装束资源中的信息，对于硬盘，把它所有的束资源安装到桌面数据库中，对于软盘，安装到桌面文件; Finder使用这个信息显示您的应用程序文件类型的图标。

您必须在束资源中为您的图标列表资源指派局部资源标识符。应确保对于定义了图标的所有文件类型，这些局部资源标识符和您在它们相应的文件参考资源中指派的局部资源标识符匹配。在软盘（以及运行较早版本系统软件的硬盘）的桌面文件中，为了避免和其它应用程序的资源冲突，Finder为您给资源指派的资源标识符 重新编号。因此，束资源必须依靠这些局部资源标识符把图标列表资源映射到它们的文件参考资源; 也就是说，，束资源使用您指派给图标列表资源的局部资源标识符，把它映射到指定了同样局部资源标识符的文件参考资源。

例如，7-19 页的程序清单 7-3 中具有资源标识符 208 的文件参考资源表明，文件类型'APPL'（SurfWriter的应用程序文件）被指派了一个局部资源标识符0。在程序清单 7-4 给出的束资源中，您可以看到局部资源标识符 0 是指派给具有资源标识符 128 的图标列表资源的。这就把这个资源定义的图标（请参阅 7-14 页的图 7-7）映射到SurfWriter的应用程序文件。程序清单 7-4 给出了图标的束资源及 7-14 页的列表 7-2 和 7-19 页的程序清单 7-3 中定义的文件参考资源。

程序清单 7-4 Rez输入格式的束资源

```
resource 'BNDL' (128, purgeable) {  /*SurfWriter bundle re-
source*/
    'WAVE',  /*SurfWriter signature*/
    0,       /*resource ID of signature resource: should be 0*/
    {
    'ICN#', {/*mapping local IDs in 'FREF's to 'ICN#' IDs*/
    0, 128,  /*'FREF' w/ local ID 0 maps to 'ICN#' res ID 128*/
    1, 129,  /*'FREF' w/ local ID 1 maps to 'ICN#' res ID 129*/
    2, 130,  /*'FREF' w/ local ID 2 maps to 'ICN#' res ID 130*/
    3, 131   /*'FREF' w/ local ID 3 maps to 'ICN#' res ID 131*/
             /*no 'FREF' with local ID 4 in this list: */
             /* TeachText's icons used for 'ttro' file type*/
        },
```

```
        'FREF', {     /*local res IDs for 'FREF's: no duplicates*/
            10, 208, /*local ID 10 assigned to 'FREF' res ID 208*/
            11, 209, /*local ID 11 assigned to 'FREF' res ID 209*/
            12, 210, /*local ID 12 assigned to 'FREF' res ID 210*/
            13, 211, /*local ID 13 assigned to 'FREF' res ID 211*/
            14, 212  /*local ID 14 assigned to 'FREF' res ID 212*/
            }
        }
};
```

请注意，在程序清单7-4 中，您还把局部资源标识符指派给束资源中的文件参考资源。这个指派是多余的，因为Finder不把这些局部资源标识符映射到其它的任何资源。对束资源中的文件参考资源指派局部资源标识符是对较早版本的Macintosh系统软件执行的，为了保持和过去的兼容性，在新版本中仍然这样做。为了和束资源的格式兼容，所以把文件参考资源标识符 指派为局部资源标识符。您可以按照您需要的方式为它们编号，但是这个特定列表中的每个局部资源标识符必须是唯一的。

在组成图标库的所有图标资源类型中，您只需要列出束资源中的图标列表资源。只要您给其它的资源指派和图标列表资源同样的资源标识符，Finder能够自动地识别并装入图标库的所有其它成员。

如果用户把其它应用程序创建的文稿图标拖拽至您的应用程序图标，而且您为这些文稿的文件类型创建了文件参考资源，Finder就能够打开您的应用程序，并传送给它文稿的名称。您应该为您的应用程序所支持的所有文件类型创建文件参考资源。不要为其它应用程序创建的文件类型提供图标资源，因为Finder将使用文稿的创建者所定义的图标资源，而不会使用您的图标资源。虽然这种文件参考资源的局部资源标识符（在文件参考资源中及在束资源的底部）是多余的，资源格式仍然要求您在这两处提供局部资源标识符。

例如，在 7-19 页的程序清单 7-3 中，具有资源标识符 212 的文件参考资源的局部资源标识符是 4，但是在 7-21 页的程序清单 7-4 中，局部资源标识符 4 并没有被指派给任何图标列表资源。这个指定文件类型'ttro'的文件参考资源是在程序清单 7-3 中创建的，其目的是当用户把 TeachText 的只读文稿拖拽至 SurfWriter 的应用程序图标时，使 Finder 打开应用程序 SurfWriter。对于这个文件类型，没有在应用程序 SurfWriter 的束资源中进行图标映射，因为 Finder 显示的图标是由应用程序 TeachText 定义的。在程序清单 7-14 的束资源中具有资源标识符 212 的文件参考资源的局部资源标识符 是14，因为资源的格式要求对于所属的所有的文件参考资源都有一个局部资源标识符。

通过在文件的 Finder 标志域中设定一个标志位，您提醒 Finder，应用程序有一个束资源。（大多数开发环境提供了简单的工具来设定束的数位。7-32 页的“使用目录文件中的 Finder 信息”一节对 Finder 标志作了说明。）

图 7-8 说明了程序清单7-4 中创建的束资源如何使用局部资源标识符把图标列表资源映射到文件参考资源。这个图说明了两个主要的概念：第一，束资源把您的应用程序和它的文稿的所有图标资源和文件参考资源联系在一起；第二，图标资源和有关它们的文件参考资源通过局部资源标识符映射联系。

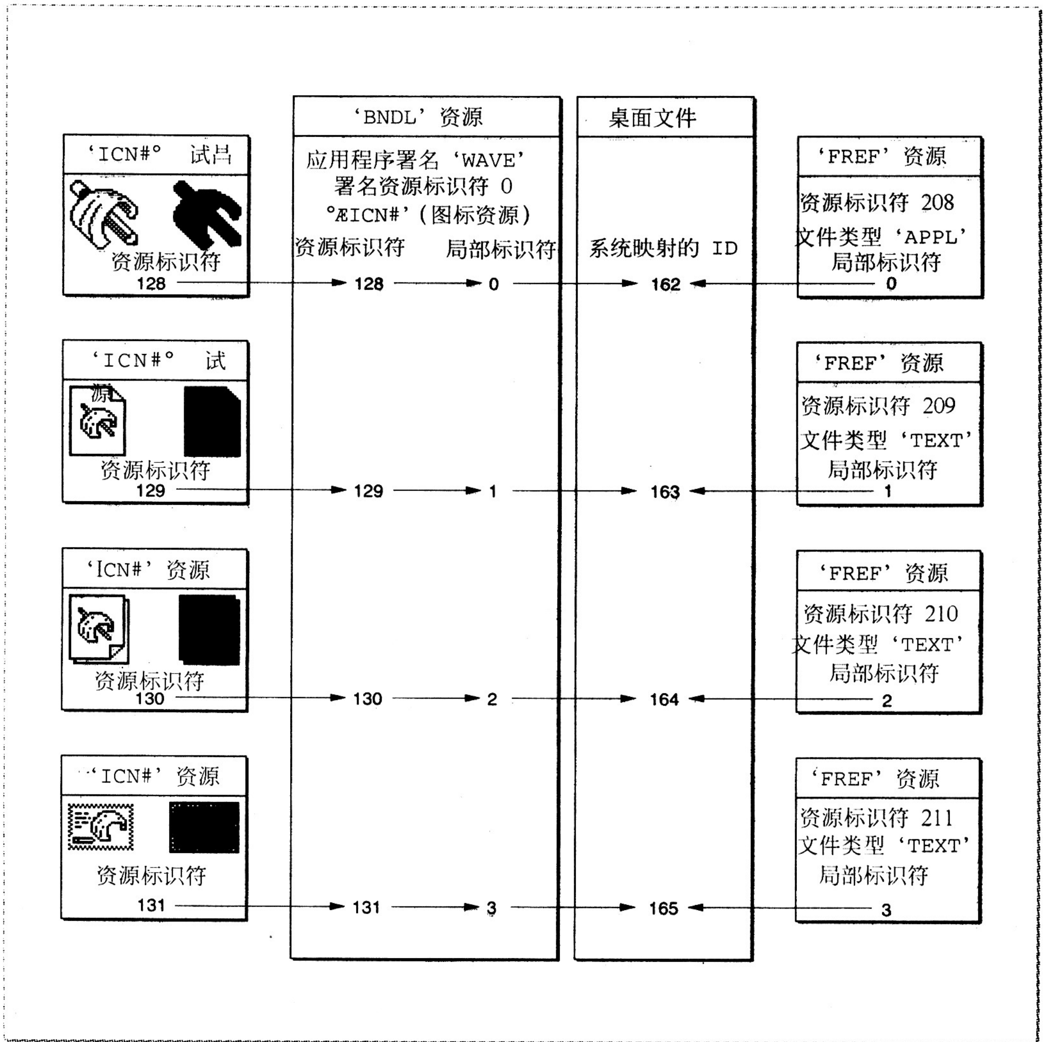


图 7-8　在束资源中链接图标列表资源和文件参考资源

在图 7-8 中，应用程序文件的图标列表资源的资源标识符是 128，而它的文件参考资源的资源标识符是 208。为了便于保留代码，您似乎应该给文件的文件参考资源和它的图标列表资源同样的资源标识符。但是，因为每当Finder把它们添加至软盘上的桌面文件时，都要为它们重新编号，所以您必须使用局部资源标识符映射它们。在图 7-8 中，应用程序文件的图标列表资源的局部资源标识符指派为 0。这样就把图标映射到具有局部资源标识符的文件参考资源所说明的文件类型——在这个例子中，文件参考资源的资源标识符是 208。

下面列举了为给应用程序和文稿提供图标您必须采取的一般步骤，这里假定您使用的是ResEdit这类工具，它允许您同时打开并编辑几个资源。（请记住，这些资源的资源标识符必须大于或等于 128。）

为了给您的应用程序和它的文稿提供图标库，应遵循下列步骤：

1. 设计一个图形元素，它供您的所有图标库共享，并可以帮助用户迅速识别和您的产品有关的文件。

2. 为您的应用程序文件创建一个图标列表（'ICN#'）资源。

3. 创建应用程序文件的图标库的其它成员，它们的资源类型是 'ics#'、'icl8'、'icl4'、'ics8' 和 'ics4'，令它们的资源标识符和图标列表资源的资源标识符一样。

4. 创建一个束（'BNDL'）资源。

5. 在束资源中，列出应用程序文件的图标列表资源的资源标识符编号，并指派它的局部资源标识符为 0。

6. 为应用程序文件创建一个文件参考资源。

7. 在文件参考资源中，指派应用程序文件的文件类型为 'APPL'，指派它的局部资源标识符为 0。

8. 在束资源中，列出应用程序文件的文件参考资源的资源标识符 编号，并给它指派一个唯一的局部资源标识符，例如 0，以便和在文件参考资源中指派的局部资源标识符保持一致。

9. 创建另一个图标库，包含资源类型'ICN#'、'cs#'、'icl8'、'icl4'、'ics8' 和 'ics4'，它们代表您的应用程序所创建的一类文稿。

10. 在应用程序的束资源中，列出文稿的图标列表资源的资源标识符 编号，并指派它的局部资源标识符为 1。

11. 为文稿创建一个文件参考资源。

12. 在文稿的文件参考资源中，指派给它一个文件类型（例如\'TEXT' 或 'edtt'），并指派它的局部资源标识符为 1。

13. 在束资源中，列出文稿的文件参考资源的资源标识符 编号，并给它指派一个唯一的局部资源标识符，例如 1，以便和在文件参考资源中指派的局部资源标识符保持一致。

14. 对于您的应用程序创建的每一类文稿，指派一个独有的局部资源标识符，重复第 9 步到第 13 步。

15. 如果您的应用程序支持其它应用程序的文件类型，请为它们定义文件参考资源，但是不要为它们创建图标资源。

16. 创建一个资源标识符为 0 的署名资源（请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节）。

17. 在文件的 Finder 标志中设定标志位 hasBundle，清除标志位 hasBeenInited。(7-32 页的“使用目录文件中的 Finder 信息”一节将对 Finder 标志进行说明。)

18. 存储并关闭所有的资源。(当您重新启动 Macintosh 电脑时，您的应用程序应该以它自己的图标出现。如果以后您对任一图标做了改变，就应清除标志位 hasBeenI-nited，并在重新启动时按下 Command - Option 键，重新建立您的桌面数据库。)

Finder 打开您的应用程序的方式和时间

本章的前一节对于 Finder 用来显示和打开您的应用程序的资源进行了说明。本节将要说明的是，当用户请求 Finder 打开您的应用程序或者打开或打印您的应用程序所支持的文稿时，Finder 如何使用前面说明的资源启动您的应用程序。

Finder 打开您的应用程序的最简单的情况就是用户双击您的应用程序图标，或者选定该图标并在 Finder 的文件菜单中选择“打开”命令。在这两种情况下，Finder 调用进程管理器启动您的应用程序。进程管理器为您的应用程序创建一个存储器分区，把您的代码装入这个分区，并且为您的应用程序设置堆栈、堆和 A5 区域［请参阅《Inside Macintosh 进程》*(Inside Macintosh: Processes)* 一书］。然后进程管理器把控制返回给 Finder。

如果您的应用程序支持基本的苹果事件［请参阅《Inside Macintosh 应用程序间通信》*(Inside Macintosh : Interapplication Communication)* 一书］，Finder 向您的应用程序发送一个打开文稿事件，然后把控制交给您的应用程序。接着，您的应用程序就执行必要的工作打开自己，例如显示一个未标题文件的文稿窗口。

当用户请求 Finder 打开或打印您的应用程序所支持的文稿时，Finder 调用进程管理器，并以同样的方式打开您的应用程序，但是您的应用程序打开或打印文稿所需要的信息是由 Finder 设置并传送给您的应用程序的。这个信息包括要打开或打印的一个文件列表。在 System 7 中，应用程序是通过苹果事件收到这个信息的，《Inside Macintosh 应用程序间通信》*(Inside Macintosh : Interapplication Communication)* 一书中已对此作了说明。

用户可以通过双击您的应用程序创建的文稿的一个图标，请求Finder打开它，用户也可以通过选定一个或多个图标，并在Finder的文件菜单中选择“打开”或“打印”，请求Finder打开或打印它们。Finder通过读每个选定的文件的创建程序域，找到文稿的创建者。通常（请参阅 7-32 页的“使用目录文件中的Finder信息”一节），您的应用程序把它的署名资源中指定的 4 字符的字符串作为它的文稿的创建者。Finder搜索其署名和每个文稿的创建者匹配的应用程序。如果文稿的创建者和您的应用程序的署名匹配，Finder就调用进程管理器，启动您的应用程序，然后把选定的一个或多个文稿的名称在一个打开文稿事件或打印文稿事件中传送给它。于是，您的应用程序就应该按照要求在一个有标题的窗口中打开这些文稿或者打印它们。[有关打开文稿的详细信息，请参阅《Inside Macintosh 文件》(Inside Macintosh: Files) 一书；有关打印文稿的详细信息，请参阅 *《Inside Macintosh 绘图》(Inside Macintosh: Imaging)* 一书。]

如果用户试图打开您的应用程序的文稿，而您的应用程序并不存在，Finder就显示一个警告栏，通知用户创建文稿的应用程序不存在。如果您为您的应用程序提供一个未找到应用程序名字字符串资源（请参阅 7-27 页的“Finder 找不到您的应用程序时显示消息”一节），Finder 就在这个警告栏中显示您的应用程序的名称。

有时，当您的应用程序正在运行时，用户可能会双击您的应用程序创建的一个文稿，这时，Finder 就向您的应用程序发送打开文稿事件。

用户还可能通过把一个或多个图标拖拽至您的应用程序图标，请求Finder启动您的应用程序。Finder 通过把文稿的文件类型（它储存在目录文件中）和您的应用程序所支持的文件类型列表进行比较，决定是否启动您的应用程序。Finder 编译您为应用程序所创建的文件参考资源中的这个列表，把这个列表储存在桌面数据库中。如果文稿的文件类型出现在您的应用程序的文件参考资源列表中，Finder 就调用进程管理器，打开您的应用程序，并在一个打开文稿事件中传送给它选定的一个或多个文稿的名称。然后您的应用程序应该在有标题的窗口中开启这些文稿。

您也可以在您的文件参考资源中指定磁盘、文件夹、和对所有其它文件的通配符文件类型，这样，用户可以通过把它们的图标拖拽至您的应用程序图标，打开您的应用程序，这时，Finder打开您的应用程序并向它发送一个打开文稿事件。一个打开文稿事件包括应用程序应该打开的对象的替身记录的一个列表。按照用户的需要打开磁盘、文件夹或所有可能的文件类型是您的应用程序的任务。（有关替身记录请参阅 7-39 页开始的“使用替身”一节。）

为了支持信笺，您的应用程序应该在它的‘SIZE’资源中指定 isStationeryAware 常数，并保持检查 Finder 传送给它的一个文稿的标志位 isStationery。如果对于用户希望打开的文件，标志位 isStationery 被设定，您的应用程序应该把固定样板文稿的内容拷贝到一个新的文稿中，并在一个无标题的窗口中打开这个文稿。请参阅 7-34 页的“支持固定样板文稿”一节。

在 System 7 中，用户可以创建替身，它们是代表其它文件、目录或卷的对象。如果用户打开一个替身，它代表您的应用程序创建的文稿，Finder 就为您分辨这个替身；即，它向您的应用程序传送文稿本身（而不是替身）的名称和位置。

Finder 找不到您的应用程序时显示消息

当用户双击一个文件或者选定它并在Finder的文件菜单中选择打开或打印命令时，Finder 寻找其署名储存在文件的创建者域中的应用程序。Finder 启动该应用程序，并通知它用户希望打开或打印的文稿。如果 Finder 找不到作为创建者的应用程序，它就显示一个警告栏。

如果文稿属于 'TEXT' 或'PICT'类型，而且应用程序 TeachText 存在，警告栏就询问用户是否可以使用应用程序 TeachText 打开文稿。对于其它任何类型的文稿，或者如果应用程序 TeachText 不存在，Finder 显示的警告栏就如图 7-9 所示。您的应用程序应该在它的文稿中储存两个字符串资源之一，使警告栏的消息比图 7-9 中的系统设定显示更有用。

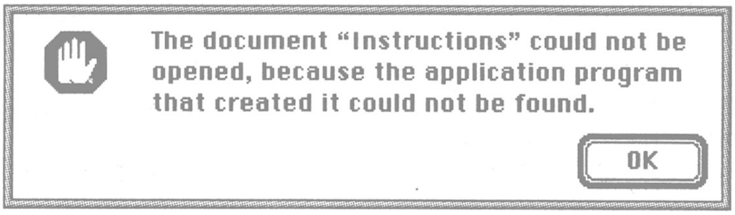


图 7-9　应用程序不存在时的系统设定警告栏

在显示图7-9 的系统设定消息之前，Finder 在文稿中寻找两个特殊的'STR' 资源之一；它们分别是未找到应用程序名字字符串和应用程序未找到消息字符串，其资源标识符分别是 -16396 和 -16397。如果 Finder 在所有安装的卷中都找不到文稿的创建程序，它就首先查找未找到应用程序信息字符串资源。如果您不想让用户打开这个文件，应提供一个未找到应用程序信息字符串资源。该消息应该解释为什么该文件不能打开。如果 Finder 没有找到一个未找到应用程序信息字符串资源，它就查找遗漏应用程序名字字符串资源。如果您想让用户打开该文件，应提供未找到应用程序名字字符串资源。未找到应用程序名字字符串应该是您的应用程序的名称Finder 在一个警告栏中显示这个字符串，通知用户通过需要您的应用程序才能打开该文件。

您应该提供未找到应用程序信息字符串资源或者未找到应用程序名字字符串，但是不应该同时提供两者。对于您的应用程序使用，而用户不应该打开的文稿（例如预置选择文件），应提供未找到应用程序信息字符串资源; 对于您认为应该让用户使用您的应用程序打开的文稿，应提供未找到应用程序名字字符串资源。

您的未找到应用程序名字字符串资源（具有资源标识符 编号 -16396 的 'STR'资源）应该包含您的应用程序的名称。程序清单 7-5 给出了应用程序 SurfWriter 的未找到应用程序名字字符串资源。

程序清单 7-5 未找到应用程序名字字符串资源的 Rez 输入格式

```
resource 'STR ' (-16396, purgeable) {/*the application name*/
     "SurfWriter"
};
```

您可以把这个资源储存在您的应用程序的资源分支中。当您的应用程序首次存储一个文稿时，它应该把未找到应用程序名字字符串资源从应用程序的资源分支拷贝到新创建的文稿的资源分支中。程序清单7-6 给出了一个应用程序定义的函数DoSaveAsCmd 的一部分，当用户在文件菜单中选择“存储为”命令时，应用程序将调用它。[有关这里用来创建、打开、和存储资源文件的文件管理器例程的说明，请参阅《Inside Macintosh 文件》*(Inside Macintosh: Files)* 一书。]

程序清单7-6 在文稿的资源分支中储存一个未找到应用程序名字字符串资源

```
VAR
     myData:        MyDocRecHnd;       {handle to document record}
     myErr:         OSErr;
     myFile:        Integer;           {file reference number}

{with the DoSaveAsCmd routine: create document's resource fork}
FSpCreateResFile(myData^^.fileFSSpec, 'MYAP', 'TEXT',
                 smSystemScript);
myErr := ResError;
IF myErr = noErr THEN {open the resource fork}
     myFile := FSpOpenResFile(myData^^.fileFSSpec, fsRdWrPerm);
IF myFile > 0 THEN      {copy the missing-application name string}
     myErr := DoCopyResource('STR ', -16396, gAppsResFile, myFile)
ELSE
     myErr := ResError;
IF myErr = noErr THEN
     myErr := FSClose(myFile);         {close the resource fork}
```

程序清单 7-7 给出了应用程序定义的函数 DoCopyResource，它把未找到应用程序名字字符串资源从应用程序的资源分支拷贝到新创建的文稿的资源分支。(有关这里用来设定、打开、和写资源文件的资源管理器例程的说明，请参阅《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 一书的“资源管理器”一章。)

程序清单7-7 把未找到应用程序名字字符串资源拷贝到文稿的资源分支

```
FUNCTION DoCopyResource (theType: ResType; theID: Integer;
                         source: Integer; dest: Integer): OSErr;
VAR
    myHandle:        Handle;  {handle to resource to copy}
    myName:          Str255;  {name of resource to copy}
    myType:          ResType; {ignored; used for GetResInfo}
    myID:            Integer; {ignored; used for GetResInfo}

BEGIN
    UseResFile(source);     {set the source resource file}
    myHandle := GetResource(theType, theID); {open the source}
    IF myHandle <> NIL THEN
    BEGIN
      GetResInfo(myHandle, myID,myType, myName);{getresource }
                                                { name}
        DetachResource(myHandle);  {detach resource}
        UseResFile(dest); {set the destination resource file}
        AddResource(myHandle, theType, theID, myName);
        IF ResError = noErr THEN
            WriteResource(myHandle);    {write resource data}
    END;
    DoCopyResource := ResError;         {return result code}
END;
```

如果当应用程序不存在时，用户试图打开或打印应用程序的一个文稿，Finder 就在警告栏中指定应用程序的名称，请参阅图 7-10。

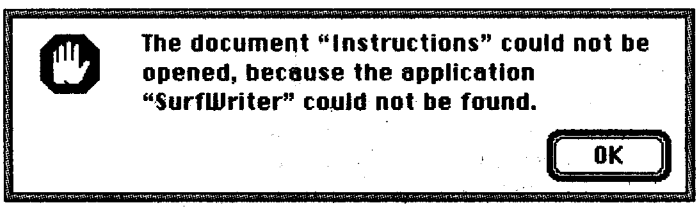


图 7-10 应用程序不存在时指定应用程序名称的警告栏

您的未找到应用程序信息字符串资源（资源标识符编号为 -16397 的 'STR' 资源）应该解释为什么用户不能打开或打印一个文稿。您应该对于不想让用户打开或打印的文件，例如您的应用程序的预置选择文件，使用这个资源。您应该登记一个不同于您的应用程序的署名（请参阅 7-8 页的"给您的应用程序一个署名，给您的文稿一个创建和文件类型"一节），并把这个署名设定为您不想让用户打开的文稿的创建者。这就保证了当用户双击这些文稿时，Finder 显示您的提示消息，而不是打开您的应用程序。

程序清单 7-8 给出了一个未找到应用程序信息字符串资源，它解释了为什么用户不能开启一个预置选择文件。

程序清单 7-8 未找到应用程序信息字符串资源的Rez输入格式

```
resource 'STR ' (-16397, purgeable) {    /*the message*/
"This document describes user preferences for the application "
"SurfWriter. You cannot open or print this document. To be "
"effective, this document must be stored in the Preferences "
"folder in the System Folder."
};
```

图 7-11 显示了程序清单 7-8 生成的警告栏。

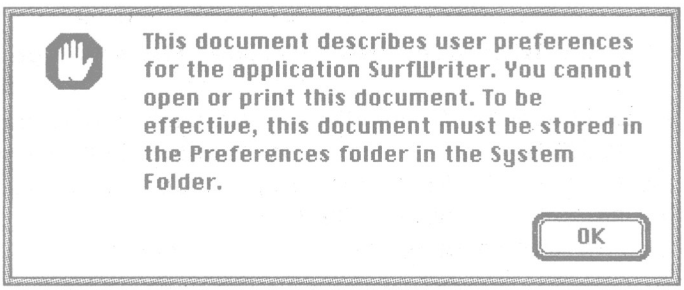


图 7-11 具有自定消息的应用程序不存在警告栏

请注意，如果您的应用程序创建文件类型为'TEXT' 或 ' PICT' 的文稿，而应用程序 TeachText 存在，若用户试图在您的应用程序不存在时从 Finder 打开这些文稿，Finder 总是显示如图 7-12 的警告栏。对于这些文件类型，即使您提供了未找到应用程序名字字符串资源或未找到应用程序信息字符串资源，Finder仍然显示这个警告栏。

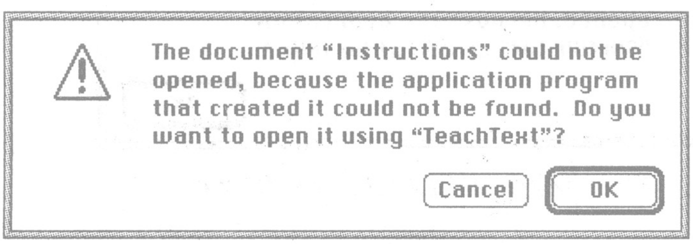


图 7-12 对于'TEXT'和 'PICT' 类型文稿的应用程序不存在警告栏

提供版本资源

您可以使用版本（'vers'）资源为您的应用程序记录版本信息。如果用户打开显示控制板，单击显示版本框，然后在显示菜单中选择“按照图标”或“按照小图标”以外的任何命令，文件名和它们的版本号就出现在激活的Finder窗口中。当用户选定您的应用程序，并在文件菜单中选择“获取信息”时，Finder还显示版本信息。

版本资源允许您储存版本号、版本消息和一个区域代码。(因为“获取信息”命令的信息窗口已经显示了您的应用程序的名称，版本消息不应该再包括它。)您可以使用版本资源把版本信息指派给单独的文件或整个文件超集(如果它是一个较大的文件集合的一部分)。资源标识符编号为 1 的版本资源指定了文件的版本，资源标识符编号为 2 的版本资源指定了文件超集的版本。

每个版本资源应该包含下列元素：

- 以二进制编码的十进制格式表示的主要修正。虽然Finder不在任何地方显示它，您可以把这个信息储存在这里；大多数编程环境提供了设定这个元素的工具
- 以二进制编码的十进制格式表示的次要修正。虽然Finder不在任何地方显示它，您可以把这个信息储存在这里；大多数编程环境提供了设定这个元素的工具
- 开发阶段。您可以使用下面这些值或代表它们的常数

值	**常数**	**说明**
0 × 20	development	alpha 预文件
0 × 40	alpha	alpha 文件
0 × 60	beta	beta 文件
0 × 80	release	正式引用文件

- 引用前修正。如果软件仍处于引用前的阶段，这个编号就指定了版本
- 区域代码。它标示这一版软件适用的语系系统。有关可以在这里指定的各种区域代码所代表的值的信息，请参阅《Inside Macintosh 文本》*(Inside Macintosh: Text)* 一书的“语系管理器”一章
- 版本编号。这个字符串标示软件的版本编号。如果用户打开显示控制板，单击显示版本框，然后在显示菜单中选择除了“按照图标”或“按照小图标”以外的任何命令，包含这个应用程序的Finder窗口就显示这个字符串

- 版本消息。这个字符串标示版本编号，以及文件的公司版权或者文件超集的产品名称。当用户选定这个文件并选择“获取信息”命令时，Finder就在信息窗口中按照如下方式显示这个字符串：
  - 对于资源标识符为 1 的版本资源，这个字符串在信息窗口的版本域中显示。
  - 对于资源标识符在 2 的版本资源，这个字符串显示在信息窗口的顶部，文件名的下面，文件图标的旁边。

程序清单7-9 给出了一个图形应用程序和它所属的文稿处理系统的版本资源。请注意，涂色程序是 1.0 版，而组成整个文稿处理系统的文件集合是 2.0 版。

程序清单 7-9 一对版本资源的Rez输入格式

```
resource 'vers' (1, purgeable) {
     0x01, 0x00, release, 0x00, verUS,
     "1.0",
     "1.0 (US), © My Company, Inc. 1992"
};
resource 'vers' (2, purgeable) {
     0x02, 0x00, release, 0x00, verUS,
     "2.0",
     "(for SurfWriter 3.0)"
};
```

图 7-13 说明了Finder如何在它的信息窗口中显示这些资源的信息。

您可以在任何种类的文件中储存版本资源，不仅仅是应用程序。如果您的应用程序不包含资源标识符为 1 的一个版本资源，Finder就把您的署名资源中的字符串作为版本信息在您的应用程序的信息窗口中显示。

使用目录文件中的 Finder 信息

每一卷中都有一个目录文件，它保留该卷中的文件和目录之间的关系。(一个卷可以是任何一种存储媒介，它被格式化后可以存储文件。) 尽管使用目录文件最多的是文件管理器，但是目录文件也包含Finder使用的信息。文件的信息是列在文件信息记录(FInfo类型的数据结构) 和扩展文件信息记录 (FXInfo类型的数据结构) 中的。目录的信息则列在目录信息 (DInfo) 记录和扩展目录信息 (DXInfo) 记录中。

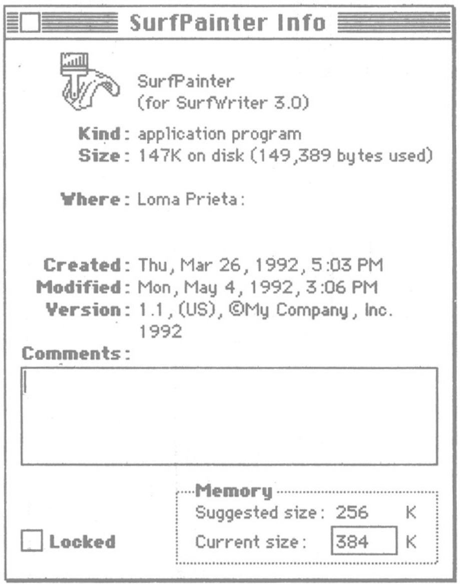


图 7-13 信息窗口中的版本数据

Finder 操纵文件信息、目录信息和扩展目录信息记录中的域；您的应用程序不应该直接检查或设定这些域。

通常，当您的应用程序创建一个新文件时，它在该文件的文件信息记录域中设定文件类型和创建者信息；例如，文件管理器函数 FSpCreate［请参阅《Inside Macintosh 文本》*(Inside Macintosh: Files)* 一书］接受创建者和文件类型作为参数。Finder 操纵文件信息记录中的其它域，下面给出了所有的域：

```
TYPE      FInfo =
          RECORD
          fdType:        OSType;  {file type}
          fdCreator:     OSType;  {file creator}
          fdFlags:       Integer; {Finder flags}
          fdLocation:    Point;   {file's location in window}
          fdFldr:        Integer; {directory that contains file}
          END;
```

在您创建一个文件后，可以使用文件管理器函数FSpGetFInfo 返回文件信息记录，然后用文件管理器的函数 FSpSetFInfo 改变 fdType 和 fdCreator 域。

您可以通过调用文件管理器的函数FSpGetFInfo或PBGetCatInfo检查这个记录中的信息。您可能希望检查文件的文件类型或创建者，或者设定您的文稿的一个Finder标志。有关所有Finder标志的列表，请参阅 7-47 页的“文件信息记录”一节。

设定的所有 Finder 标志说明如下:

- isInvisible。该标志指定了这个文件在Finder对话框或标准文件软件包对话框是不可见的。在一般情况下，使一个文件不可见不是一个好方法。因为包含临时文件的临时项目文件夹是不可见的，所以即使是临时文件也不需要是不可见的。有关临时项目文件夹请参阅 7-41 页的“使用系统文件夹及和它有关的目录”一节
- hasBundle。该标志指定了一个文件有束资源，它把文件和您自己的图标联系起来。当Finder显示或操纵一个文件时，它检查文件的hasBundle标志位。如果该数位没有被设定，Finder就为这种文件类型显示系统设定图标。如果hasBundle标志位被设定，Finder 就检查 hasBeenInited 标志位。如果 hasBeenInited 标志位被设定，Finder 就使用桌面数据库中的信息显示该文件的图标。如果hasBeenInited标志位没有设定，Finder就从束资源把信息安装到桌面数据库中，并设定 hasBeenInited 标志位。当您创建应用程序时，大多数开发环境都提供了一个设定标志位的简单工具
- nameLocked。该标志指定了一个文件不能在 Finder 中重新命名，用户也不能给这个文件自定图标
- isStationery。该标志指定了一个文件是固定样板文稿。为了支持固定样板文稿，您的应用程序应该对Finder或标准文件软件包传送给它的每一个文稿检查这个标志位。（文件管理器函数 StandardGetFile 和 CustomGetFile 在标准文件回答记录的sfFlags域返回这个标志。）如果用户希望打开的一个文件的isStationery数位被设定，您的应用程序应该把模板的内容拷贝到一个新文稿，并在一个未标题文件的窗口中打开它。下一节将对固定样板文稿进行说明
- isShared。该标志指定了一个文件是应用程序，网络上的多个用户可以同时执行它
- hasCustomIcon。该标志指定了有一个自定图标。7-17 页的“创建自定文稿图标”一节解释了用户和您的应用程序应该如何使用自定图标

支持固定样板文稿

固定样板文稿是用户作为模板创建的特殊文稿。打开一个固定样板文稿并不是打开文稿本身，而是打开一个和固定样板文稿有同样内容的一个新文稿。把任何文稿转换成一个固定样板文稿的步骤是：用户选定一个文稿，在文件菜单中选择“获取信息”，然后在信息窗口中单击固定样板文稿复选框。Finder通过在文件的Finder标志域中设定 isStationery 标志位，标示一个文稿是固定样板文稿。

当用户从 Finder 打开一个固定样板文稿时，Finder 首先检查您的应用程序的大小资源，确定它是否支持固定样板文稿。'SIZE' 资源告诉 Finder 和进程管理器，您的应用程序支持哪些功能，以及它启动您的应用程序时应分配多大的存储器。列表 7-10 给出了一个大小资源。

程序清单 7-10 大小资源的 Rez 输入格式

```
resource 'SIZE' (-1, purgeable)  {
     reserved,
     acceptSuspendResumeEvents,
     reserved,
     canBackground,
     doesActivateOnFGSwitch,
     backgroundAndForeground,
     dontGetFrontClicks,
     ignoreAppDiedEvents,
     is32BitCompatible,
     isHighLevelEventAware,
     localAndRemoteHLEvents,
     isStationeryAware,              /*support stationery pads*/
     dontUseTextEditServices,
     reserved, reserved, reserved,
     kPrefSize * 1024,
     kMinSize * 1024
};
```

请注意第 12 个域 isStationeryAware，它告诉 Finder 这个应用程序支持固定样板文稿。

如果署名资源中的 isStationeryAware 标志位没有设定，Finder 就从该模板创建一个新文稿，并向用户告诫其名称。然后，Finder 和平常一样启动您的应用程序，传送给它新文稿的名称。

如果 isStationeryAware 标志位被设定（如程序清单 7-10 所示），Finder 就通知您的应用程序，用户已经打开了一个文稿，并向您的应用程序传送固定样板文稿的名称。

若要支持固定样板文稿，您的应用程序必须:

- 在它的署名资源中指定 isStationeryAware 常数
- 在打开每一个文稿前，都要检查它的 isStationery 标志位

7-36 页的程序清单 7-11 给出了一个简单的函数，它接受文件系统规格说明记录，并返回 TRUE 或 FALSE，指示该文件是否固定样板文稿。

程序清单 7-11 确定一个文稿是否固定样板文稿

```
FUNCTION IsStationeryDoc (myFSSpec: FSSpec): Boolean;
VAR
    myErr:      OSErr;
    myFInfo:    FInfo;
BEGIN
    myErr := FSpGetFInfo(myFSSpec, myFInfo);
    IF myErr = noErr THEN
     IsStationeryDoc := BTST(myFInfo.fdFlags, isStationery)
    ELSE
        IsStationeryDoc := FALSE;
END;
```

isStationery标志位本身就标示了一个文稿是否固定样板文稿。如果 isStationery标志位对一个用户希望打开的文件设定，您的应用程序应该把模板的内容拷贝到一个新文稿，并在一个未标题文件的窗口中显示它。[有关打开文稿和文件管理器函数 FSpGetFInfo 的信息，请参阅《Inside Macintosh 文件》*(Inside Macintosh: Files)* 一书。]

您的应用程序可以检查标准文件回答记录的sfFlags域，确定isStationery 标志位是否设定。标准文件软件包和Finde不一样，无论isStationery是否设定，它向您的应用程序传送的是固定样板文稿本身，而不是它的一个拷贝。当用户在您的应用程序中打开一个固定样板文稿时，标准文件软件包检查您的应用程序的大小资源。如果您的应用程序不支持固定样板文稿，标准文件软件包就显示一个警告栏，警告用户将要打开的是固定样板文稿本身，而不是它的一个拷贝。您可以看到，用户仍然可以容易地改变模板，并通过不指定新的名称选择“存储”而重写它。您可以使应用程序对于固定样板文稿显示特别警告，防止这种使用户懊恼的事情发生。

在您的文稿中，应告诉用户可通过选择“获取信息”命令来制做固定样板文稿。您也可能希望给出您的应用程序所创建的有用的固定样板文稿的实例。例如，如果您的应用程序支持文字和图形，您可以提供有关业务信函或业务帐单的示例。

分配字体、语音及其它可移动资源

如果您创建字体、语音、键盘布局和系统资源集合，您可以在单独的、可移动的资源文件中分配它们。

可移动资源，例如字体、键盘布局和语音在屏幕上由图标代表。为了安装这些资源，用户把它们的图标拖拽至系统文件夹图标。Finder 把字体资源放入字体文件夹，把其它资源放入系统文件。用户可以通过双击系统文件夹，把它打开，然后双击字体文件夹，以便确定当前安装的字体。通过双击系统文件，使它像文件夹一样打开，用户就可以看到所安装的其它可移动资源。(有关系统文件夹的新的组织的说明，请参阅 7-41 页的“使用系统文件夹及和它有关的目录”一节。)

为了使这些资源在屏幕上可见，应对每个资源指派一个 Finder 为可移动资源定义的特殊文件类型。下面的列表给出了可移动的资源，它们被指派的文件类型及它们的图标:

资源	**文件类型**	**大黑白图标**
字体	'ffil'	
键盘布局	'kfil'	
语系系统 资源集合	'ifil'	
语音	'sfil'	
TrueType 字体	'tfil'	

附注:

只要文件不是安装在系统文件或手提式文件中，您或您的用户可以为这些文件类型提供自定图标（请参阅 7-17 页开始的“创建自定文稿图标”一节)。只要文件被安装在系统文件或手提式文件中，Finder 就用上面列表中的图标显示它们。即使字体和 TrueType 字体可移动资源被安装在字体文件夹中，也保留它们的自定图标。

用户仍然可以把字体（以及桌面附件）储存在具有手提式图标的文件中，这正是在 System 7以前版本的系统软件中，分配安装它们或者用户使用 Font/DA （字体／桌面附件）移动程序存储它们的方法。存储桌面附件的手提式文件属于'DFIL'类型，存储字体的手提式文件属于'FFIL' 类型。所有手提式文件的创建者是'DMOV'。

在您的应用程序中，应告诉用户通过把字体、语音或语系系统资源集合的图标拖拽至系统文件夹图标，安装它们。这时，将出现一个对话框，要求用户确认这些资源是否应该装入字体文件夹或系统文件夹。用户通过单击“确认”接受安装。用户还可以选择“取消”，不作安装。

附注:

如果用户把图标拖拽至打开的系统文件夹窗口，而不是系统文件夹图标，Finder 就把文件拷贝或移动到系统文件夹目录中，而不是把它们自动识别地安装在字体文件夹或系统文件夹中。

为非文稿图标提供旁述

Finder 为用户提供联机旁述帮助。用户在帮助菜单中选择“显示旁述”后，把光标移至和帮助资源有关的屏幕的一个区域（例如一个菜单、窗口控制部件或对话框），就会出现说明性的旁述。

Finder 为应用程序、控制板和系统功能扩展图标提供了系统设定旁述。您可以在应用程序的资源分支中添加一个资源标识符为 -5696 的 'hfdr' 资源，为您的应用程序、控制板和系统功能扩展图标提供自定旁述。图 7-14 把 SurfWriter 的应用程序图标的系统设定旁述和自定旁述进行了比较。

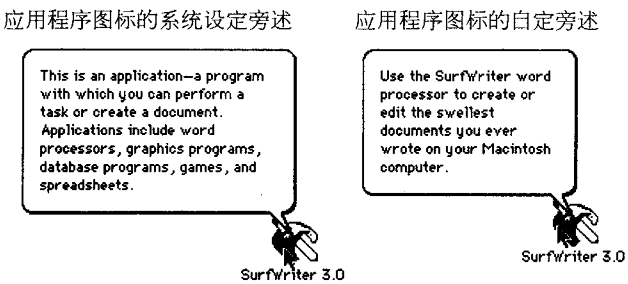


图 7-14　应用程序图标的系统设定和自定旁述

程序清单 7-12 给出了一个超越 Finder 帮助的资源及和它有关的 'STR' 资源，它们被用于图 7-14 中显示的自定旁述。

附注:

您不能超越 Finder 对文稿图标使用的系统设定旁述。

《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 一书的“帮助管理器”一章详细说明了如何为您的应用程序图标和其它元素提供旁述。

程序清单7-12 应用程序图标的旁述资源的Rez输入格式

```
resource 'hfdr' (-5696, purgeable) {         /*help for SurfWriter icon*/
     HelpMgrVersion, hmDefaultOptions, 0, 0, /*header information*/
     {HMSTRResItem {kIconHelpString}}
};
resource 'STR ' (kIconHelpString, purgeable) {/*help message for app icon*/
     "Use the SurfWriter word processor to create or edit the "
     "swellest documents you ever wrote on your Macintosh computer."
};
```

使用替身

Finder允许用户创建多个图标代表一个文稿或其它桌面对象（例如磁盘、文件夹或废纸篓）。只有一个图标代表真实的文件，其它都是指向原文件的替身。替身是代表其它某个文件、目录或卷的一个对象。替身的外观就像它的目标的图标，但是其名称以不同的字形显示。所采用的字形取决于系统的语系，对于Roman和大多数其它语系，替身的名称用斜体显示。

对于用户而言，真实文件的图标和它的替身在功能上完全相同。替身使用户在组织文件时有更大的灵活性，并为储存文件服务器中的大文件或动态文件的本地拷贝提供了方便。

通常，当用户希望打开或者打印文件时，您的应用程序不需要关心它们是否为替身，因为 Finder和标准文件软件包都会在把它们传送给您的应用程序前，进行分辨。如果用户打开您的应用程序所创建的一个文稿的替身，Finder会向您的应用程序直接传送文稿本身，而不是替身的名称和位置。同样，当用户在您的应用程序中打开一个替身时，标准文件软件包向您的应用程序传送目标文稿的名称。

如果您的应用程序不通过Finder或标准文件软件包打开一个文件或一个目录（例如，它使用预置选择文件或字典文件），您的应用程序总是应该在将打开文件前，调用ResolveAliasFile函数。

替身作为一个Finder对象，描绘了一个文件（称为替身文件），它包含一个指向该图标代表的文件、目录或卷的记录。替身文件是由用户通过Finder创建并管理的。

虽然您的应用程序不应该创建替身文件或改变用户的替身文件，但是它可以创建并使用自己的替身记录，以便储存文件或目录的识别信息。替身记录是一个数据结构，它识别一个文件、文件夹或卷。在您的应用程序下一次需要文件和文稿时，即使用户已经对它们进行了各种操作，如重新命名、复制、从备份中还原或移动，您仍然可以使用替身管理器确定它。您还可以使用替身记录识别其它卷的对象，包括AppleShare卷。有关在替身记录中创建和管理信息的细节，请参阅《Inside Macintosh 文件》*(Inside Macintosh: Files)* 一书的“替身管理器”一章。

替身文件包含一个替身记录，它作为 'alis' 资源类型储存，指向替身的目标。(替身目标是替身记录所说明的文件、目录或卷。)替身文件还可以包含目标对象的图标说明。Finder通过在文件的Finder标志域设定isAlias标志位识别一个替身文件(有关 Finder 标志请参阅 7-47 页的“文件信息记录”一节)。

通常，一个代表文稿的替身文件和它代表的文件有同样的类型和创建者。但是，许多Finder对象，例如磁盘、文件夹和废纸篓，并没有文件类型。因此，这些对象的替身文件被指派特殊的文件类型，称为替身类型。下面是用户可以创建替身的那些对象的替身类型:

对象	**替身类型**	**常数**
苹果菜单命令项目文件夹	‘faam’	kAppleMenuFolderAliasType
AppleShare 文件夹	‘fadr°	kDropFolderAliasType
应用程序	‘adrp’	kApplicationAliasType
控制板文件夹	‘fact’	kControlPanelFolderAliasType
输出		
AppleShare文件夹	‘faet’	kExportedFolderAliasType
功能扩展文件夹	‘faex’	kExtentionFolderAliasType
文件服务器	‘srvr’	kContainerServerAliasType
软盘	‘flpy’	kContainerFloppyAliasType
文件夹	‘fdrp’	kContainerFolderAliasType
硬盘	‘hdsk’	kContainerHardDiskAliasType
安装的		
AppleShare文件夹	‘famn’	kMountedFolderAliasType
能够保持文件的其它对象	‘drop’	kContainerAliasType
预置选择文件夹	‘fapf’	kPreferencesFolderAliasType
PrintMonitor文稿文件夹	‘fapn’	kPrintMonitorDocsFolderAliasType
共享AppleShare文件夹	‘fash’	kSharedFolderAliasType
启动项目文件夹	‘fast’	kStartupFolderAliasType
系统文件夹	‘fasy’	kSystemFolderAliasType
废纸篓	‘trsh’	kContainerTrashAliasType

(有关下列内容请参阅 7-41 页的“使用系统文件夹及和它有关的目录”一节: 功能扩展、预置选择、苹果菜单命令项目、控制板、启动项目和打印监控程序文稿文件夹。)

当不通过Finder或标准文件软件包打开一个文件时，您应该在即将打开文件前调用ResolveAliasFile。(有关ResolveAliasFile函数请参阅 7-52 页。)在程序清单7-13 中，自定的打开函数 MyOpen保证了要打开的文件是目标文件，然后再用文件管理器的函数FSpOpenDF打开数据分支。

程序清单 7-13 使用ResolveAliasFile 函数打开文件

```
FUNCTION MyOpen (VAR theSpec: FSSpec; perm: SignedByte;
                    VAR fRefNum: Integer): OSErr;
VAR
    myErr:                  OSErr;
    targetIsFolder:         Boolean;
    wasAliased:             Boolean;
BEGIN
    myErr := ResolveAliasFile(theSpec, TRUE, targetIsFolder, wasAliased);
    IF targetIsFolder THEN
        myErr := paramErr           {cannot open a folder}
    ELSE IF (myErr <> noErr ) THEN      {try to open it}
        myErr := FSpOpenDF(theSpec, perm, fRefNum);
    MyOpen := myErr;
END;
```

使用系统文件夹及和它有关的目录

系统文件夹是一个目录，它主要储存系统软件，例如系统文件、Finder和打印机驱动程序。System 7 为系统文件夹引入了一个新的组织，它包含一组新的子目录来容纳有关的文件。Finder使用这些子目录帮助用户方便地管理文件。例如，通过为用户把桌面附件、控制板、字体、预置选择文件、系统功能扩展和临时文件分类并储存在各自的文件夹中，Finder避免了在系统文件夹的顶层杂乱无章地堆积几十个甚至几百个文件。

用户可以通过把字体、语音、键盘布局、控制板和系统功能扩展的图标拖拽至系统文件夹图标，方便地安装它们。然后，Finder把它们移至适当的子目录。例如，当控制板图标被拖拽至系统文件夹图标时，Finder 给出一个对话框，询问用户“把这个控制板放入‘控制板’文件夹？”用户单击“确认”按钮表示同意，单击“取消”按钮表示否定。

附注:

如果用户把图标拖拽至打开的系统文件夹窗口，而不是系统文件夹图标，Finder就把文件拷贝或移动到系统文件夹的目录中，而不是把它们拷贝或移动到适当的子目录中。

图 7-15 显示了用户通常在系统文件夹中看到的新组织的目录

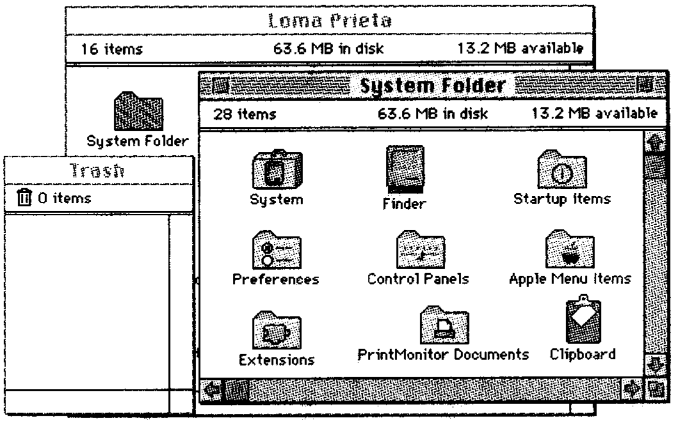


图 7-15　　系统文件夹和有关的文件夹

其它有关的目录位于根目录内。请注意废纸篓窗口。它显示了废纸篓目录的内容，它向用户显示的是废纸篓图标。废纸篓目录处在卷的根目录层。网络环境中的 Macintosh 用户共享文件在一个共享的废纸篓目录中保留了各自的废纸篓子目录。也就是说，服务器为每个在Macintosh服务器上打开一个卷，并把对象拖拽至废纸篓图标的用户，创建单独的、唯一命名的废纸篓子目录。用户不能看到共享废纸篓目录中的任何废纸篓子目录。在桌面环境中，用户只能看到本地 Macintosh 电脑上的废纸篓图标。当用户双击这个废纸篓图标时，在窗口中只显示用户已经扔掉的文件的名称，而不为用户标明这些文件来自哪个电脑。

在卷的根目录层，Finder 还保留一个临时项目文件夹和一个桌面文件夹，用户看不到它们，所以在图 7-15 中没有出现。

图 7-15 显示的是单用户系统中文件夹一般的组织情况。在所有显示的有关目录中，您的应用程序可能只会使用预置选择文件夹和临时项目文件夹。但是，您无法确定它们或其它与系统有关的目录的位置。以后，这些与系统有关的目录可能不再位于系统文件夹或根目录中。

您可以使用 FindFolder 函数（请参阅 7-54 页）获得这些目录的路径信息。在这些目录中，您需要的可能只有预置选择、临时项目和废纸篓目录。例如，您可能希望执行以下任务：检查用户的一个配置文件是否在预置选择文件中，在临时项目中创建一个临时文件，或者当您试图储存一个文件而应用程序的存储盘区用完时，检查废纸篓目录的项目占用了多少存储，并向用户报告。

您的应用程序可以自由地使用下面两个目录储存重要的文件并确定它的位置:

- **预置选择,** 它位于系统文件夹中，保持着记录局部配置设定状态的预置选择文件。您的应用程序可以在这个目录中储存自己的预置选择文件。激活的Finder 预置选择文件总是储存在预置选择文件夹中的。不应该在预置选择文件夹中储存网络上一个以上Macintosh 电脑的用户共享的信息。您应该确保应用程序在它的预置选择文件被删除的情况下也能运行

- **临时项目**，它位于卷的根目录，保持着应用程序创建的临时文件。临时项目文件夹对于用户是不可见的。您的应用程序可以把它的临时文件放置在这个目录中。只有当您的应用程序需要保持临时文件打开时，它才应该存在。只要您的应用程序关闭该临时文件，就应该删除它。您还应该确保为您的临时文件指派的名称是唯一的，这样就不会重写另一个应用程序的文件

您应该牢记有关储存应用程序文件的几个规则。

首先，不要在系统文件夹的顶层储存任何文件。应使用预置选择目录或在下面列表中说明的一个目录。

第二，应使用 FindFolder 函数确定文件的位置或把文件放入正确的位置。不能假定文件是和您的应用程序在同一个卷中；它们完全可能在不同的本地卷中，也可能在网络的远程卷中。

第三，不要在预置选择目录或系统文件夹的任何一个目录中储存多个用户需要存取的任何文件。请记住，系统文件夹中的文件一般只能由在该系统文件夹中启动系统文件的个人存取。

用户和 Finder 还能使用其它目录储存重要的文件并确定它的位置；这些目录在下面列出。在一般情况下，您的应用程序不应该在这些目录中储存文件:

- **苹果菜单命令项目,** 它位于系统文件夹中，保持着标准的桌面附件和其它所有桌面附件、应用程序、文件、文件夹或用户希望在苹果菜单中显示的替身。只有用户和安装程序可以把文件放入苹果菜单命令项目文件夹

- **控制板,** 它位于系统文件夹中，保持着控制板。苹果菜单命令项目文件夹保持着一个控制板文件夹的替身，这样用户也可以通过苹果菜单存取控制板。只有用户和安装程序可以把文件放入控制板文件夹

- **桌面文件夹,** 用户看不到它。它位于卷的根目录。桌面文件夹储存出现在屏幕的桌面上的图标的有关信息。用户通过在屏幕上排列图标，控制桌面文件夹的内容。对于用户而言，出现在屏幕上的是所有已安装卷的桌面文件夹的内容的和

- 功能扩展，它位于系统文件夹中，保持着功能扩展文件，即这些代码不是基本系统软件的一部分，但是它们提供系统级的服务，例如打印机驱动程序和系统功能扩展。所有类型为 'INIT' 的文件（以前称为启动文稿，类型为 'appe'，也被称为仅限后台使用的应用程序（没有用户界面））被Finder送到这个文件夹。类型为'scri'的文件（语系系统的系统功能扩展）也被送到这个文件夹。只有用户和安装程序可以把文件放入功能扩展文件夹

- 字体，它位于系统文件夹中，其电脑的系统软件必须是 7.1 或以上版本，它保持着字体。只有用户和安装程序可以把字体放入字体文件夹

- 打印监控文稿，它位于系统文件夹中，保持着等待打印的假脱机文稿。只有打印软件使用打印监控程序文稿文件夹

- 来自卷名的拯救项目，它位于废纸篓目录中，只有当Finder在系统启动、重新启动或关机时，在临时项目文件夹中找到项目，才创建这个目录。因为应用程序应该在关闭临时文件时删除它们，所以，如果Finder在临时项目文件夹中发现一个文件，就表明系统崩溃。当Finder在临时项目文件夹中发现一个文件时，它就创建一个目录，称为来自卷名的拯救项目目录，它是对于临时项目文件夹所在的卷命名的。例如，当Finder在名为Loma Pieta 的卷中的临时项目文件夹中发现一个文件时，它创建一个来自Loma Prieta的拯救项目。然后，Finder把临时文件移至该目录中，如果用户希望继续执行因系统崩溃而中断的任务，就可以考察这个文件。当用户倒空废纸篓时，所有拯救项目文件夹都消失。只有Finder可以把文件放入拯救项目目录

- 启动项目，它位于系统文件夹中，保持着应用程序和桌面附件（或它们的替身）
- 它们是用户希望每当Finder启动时，也启动的应用程序和桌面附件。只有用户才可以把文件放入启动项目文件夹。请注意，用户放入启动项目文件夹的启动应用程序和文件类型为'INIT的系统功能扩展（原来称为启动文稿）有所不同，后者通常是储存在功能扩展文件夹中的

- 系统文件，它位于系统文件夹中，保持着基本系统软件以及一些系统资源，例如语音和键盘资源。在下述意义上系统文件就像一个文件夹：尽管它像一个手提式图标，但是只要双击它就打开一个窗口，它展示储存在系统文件中的可移动资源文件（例如语音、键盘布局和语系系统资源的集合；请参阅 7-36 页的“分配字体、语音和其它可移动资源”中有关可以移至系统文件的资源的说明）。只有用户和安装程序可以把资源放入系统文件

- 废纸篓，位于卷的根目录的层次，保持着用户移至废纸篓图标的项目。在打开废纸篓图标后，用户看到他或她移至废纸篓图标的所有项目的集合，即来自所有已安装卷的所有适当废纸篓目录的和。在网络环境中用户间共享文件的Macintosh设置，为远程用户在它共享的废纸篓目录中保留单独的废纸篓子目录。也就是说，服务器为每个在Macintosh文件服务器上打开一个卷，并把对象拖拽至废纸篓图标的用户，创建单独的、唯一命名的废纸篓子目录。用户不能看到共享废纸篓目录和其中的任何废纸篓子目录。只有当一个目录的用户选择“清倒废纸篓”命令时，Finder才倒空这个废纸篓目录（如果是文件服务器，就是一个废纸篓子目录）

尽管和系统有关的文件夹的名称在不同的国际系统中是不同的，临时项目和桌面文件夹这两个不可见目录在所有系统中的名称都一样。系统软件为不可见的废纸篓子目录指派独有的名称。

一般情况下，您应该在应用程序的文件夹中储存应用程序专用的文件，不应该把它们储存在上述这些和系统有关的目录中。您的应用程序可能希望为用户提供一种指定目录的方法，使得用户可以在这个目录中显示辅助文件。例如，您可以设计一个自定的打开文件对话框，它允许用户指定存储路径。这个技术对于查找几个应用程序共享的文件很有用。您还可以使用替身管理器跟踪文件的位置，有关细节请参阅《Inside Macintosh 文件》*(Inside Macintosh : Files)* 一书的“替身管理器”一章。

当您设计应用程序时，从用户的观点考虑您所提供的工具是很重要的。在大多数情况下，您将希望把应用程序设计成用户只处理一个代表应用程序提供的全部功能的图标。通过简化您的产品的安装和维护，从而简化了用户的使用。如果您提供具有自己图标的可选工具，例如一个目录或同义字库，最好应该允许这些工具在文件系统的任何位置都能工作，而不应该限制它们必须储存在系统文件夹中。

桌面数据库

为了迅速存取所需要的资源，Finder 保留着有关卷中的文件和目录信息的中央桌面数据库。每当添加、移动、重新命名或删除应用程序时，Finder 就更新数据库。

在正常情况下，您的应用程序不需要使用桌面数据库中的信息，也不需要使用桌面管理器例程操纵它。您的应用程序需要做的是令 Finder 操纵桌面数据库并处理下面一类桌面管理器的任务：当用户双击应用程序的图标时打开它，保留用户与文件有关的备注，以及管理应用程序使用的图标。

如果您发现有必要从桌面数据库中析取信息，甚至需要从您的应用程序出发改变桌面数据库，您应该用所提供的桌面管理器例程完成这些任务。尽管您的应用程序可能完全不需要使用它们，为了介绍的完整，《Inside Macintosh 工具箱扩展》*(Inside Macintosh: More Macintosh Toolbox)* 一书仍然对此进行了说明。

桌面数据库中的大多数信息来自卷中应用程序和其它文件的束资源。(有关设定应用程序的束位，使它的束资源储存在桌面数据库中的讨论，请参阅 7-32 页的“使用目录文件中的Finder信息”一节。) 桌面数据库包含了所有的图标定义和它们所表示的文件类型。它列出了每个应用程序可以打开的所有文件类型，以及被列为一个文件创建者的应用程序的所有拷贝或版本。桌面数据库还列出了每个应用程序在磁盘中的位置和用户在信息窗口对桌面对象所添加的备注。

Finder 为每个容量大于 2 MB 的卷，例如硬盘，保留一个桌面数据库，数据库就储存在该卷中。对于不包含自己的桌面数据库的只读卷，例如某些光盘，桌面管理器就创建它，并把它储存在启动驱动器的系统文件夹中。

为了和系统软件的旧版本兼容，Finder在一个资源文件，而不是数据库中为容量小于 2 MB 的可推出卷保留信息。

Finder 界面参考资料

本节说明Finder专用的数据结构、例程和资源。

在“数据结构”中给出了下列记录的数据结构：文件信息记录、扩展的文件信息记录、目录信息记录和扩展的目录信息记录。在“例程”中说明了分辨替身文件和查找与系统有关的文件夹的例程。在“资源”中说明了您为您的文件提供的资源，使得Finder能够把有关它们的信息转送给用户。

数据结构

每一卷中都有一个目录文件，以便保留该卷中的文件和目录之间的关系。(一个卷可以是任何一种存储媒介，它被格式化后可以存储文件。) 尽管目录文件主要供文件管理器使用，但是它也包含Finder使用的信息。文件的信息是列在文件信息记录和扩展文件信息记录中的；目录的信息则列在目录信息记录和扩展目录信息记录中。

通常，当您的应用程序创建一个新文件时，它在该文件的文件信息记录域中设定文件类型和创建者信息。[有关文件管理器和可用于创建文件的函数的详细讨论请参阅《Inside Macintosh 文件》*(Inside Macintosh: Files)* 一书。] Finder 操纵文件信息记录中的其它域。您可以通过调用文件管理器的函数 FSpGetFInfo 或 PBGetCatInfo 检查这个记录中的信息。您可能希望检查文件的文件类型或创建者，或者检查或设定您的文稿的一个 Finder 标志。

Finder操纵扩展文件信息、目录信息和扩展目录信息记录中的域；您的应用程序不应该直接检查或设定所有这些域。为了完备起见，数据结构说明如下。

文件信息记录

通常在创建文件时，您要设定文件类型和创建者信息；例如，您向文件管理器函数 FSpCreate 传送创建者和文件类型作为它的参数。Finder 操纵文件信息记录中的其它域，它们属于数据结构类型 FInfo。在您创建一个文件后，可以使用文件管理器函数 FSpGetFInfo 返回文件信息记录，然后用文件管理器的函数 FSpSetFInfo 改变 fdType 和 fdCreator 域。

```
TYPE    FInfo =
        RECORD
            fdType:       OSType;  {file type}
            fdCreator:    OSType;  {file creator}
            fdFlags:      Integer; {Finder flags}
            fdLocation:   Point;   {file's location in window}
            fdFldr:       Integer; {window that contains file}
        END;
```

域说明

fdType 文件类型。有关文件类型的讨论，请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节。

fdCreator 创建该文件的应用程序的署名。有关创建者的讨论，请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节。

fdFlags Finder 标志。您的应用程序需要设定的标志只有几个；它们已在 7-32 页的“使用目录文件中的 Finder 信息”一节中说明过。为了完备起见，这里列出了所有的 Finder 标志。

标志名称	数位编号	说 明
isAlias	15	对于一个文件，这个标志位指示，该文件是一个替身文件。对于目录，这个标志位保留，这时应设定为 0。
isInvisible	14	从Finder和标准文件软件包对话框看不到该文件和目录。
hasBundle	13	对于一个文件，这个标志位指示，该文件包含一个束资源。对于目录，这个标志位保留，这时应设定为 0。
nameLocked	12	从 Finder 中不能重新命名该文件或目录，它的图标也不能改变。
isStationery	11	对于一个文件，这个标志位指示，该文件是一个固定样板文稿。对于目录，这个标志位保留，这应设定为 0。

标志名称	数位编号	说 明
hasCustomIcon	10	该文件或目录包含一个自定图标。
保留	9	保留，设定为 0。
hasBeenInited	8	Finder 已经从该文件的束资源把信息记录到桌面数据库中，并在桌面环境中给了该文件或目录一个位置。
hasNoINITS	7	该文件不包含 'INIT' 资源；设定为 0。对于目录，这个标志位保留，应设定为 0。
isShared	6	该文件是一个应用程序，可以被多个用户同时执行。只对应用程序设定；否则设定为 0。
requiresSwitchLaunch	5	在 System 7 中不使用，设定为 0。
colorReserved	4	在 System 7 中不使用，设定为 0。
color	1-3	彩色编码的 3 个标志位。
isOnDesk	0	在 System 7 中不使用，设定为 0。

您可以使用以下常数作为这些标志的屏蔽参数：

常数

```
fHasBundle      = 8192;       {set if file has a bundle }
                              { resource}
fInvisible      = 16384;      {set if icon is invisible}
kIsOnDesk       = $1;         {unused and reserved in }
                              { System 7}
kColor          = $E;         {three bits of color }
                              { coding}
kIsShared       = $40;        {file can be executed by }
                              { multiple users }
                              { simultaneously}
kHasBeenInited= $100;         {file info is in desktop }
                              { database}
kHasCustomIcon= $400;         {file or directory has a }
                              { customized icon}
kIsStationery = $800;         {file is a stationery pad}
kNameLocked     = $1000;      {file or directory can't }
                              { be renamed from Finder, }
                              { and icon can't be }
```

```
        kHasBundle    = $2000;     { changed}
                                   {file has bundle resource}
        kIsInvisible  = $4000;     {file or directory is }
                                   { invisible from Finder & }
                                   { from Standard File }
                                   { Package dialog boxes}
        kIsAlias = $8000;          {file is an alias file}
```

fdLocation　在窗口局部坐标中指定的文件图标在窗口中的位置。

fdFldr　文件图标出现的窗口；这个信息只对 Finder 有意义。

扩展的文件信息记录

Finder 操纵扩展的文件信息记录中的域，它们属于 FXInfo 数据结构类型。您的应用程序不应该直接检查或设定所有这些域。

```
TYPE    FXInfo =
        RECORD
            fdIconID:   Integer;    {icon ID}
            fdUnused:   ARRAY[1..3] OF Integer;
                                    {unused but reserved 6 bytes}
            fdScript:   SignedByte; {script flag and code}
            fdXFlags:   SignedByte; {reserved}
            fdComment:  Integer;    {comment ID}
            fdPutAway:  LongInt;    {home directory ID}
        END;
```

域说明

fdIconID 文件图标的一个资源标识符编号，标识图标的编号是由 Finder 指派的。

fdUnused 保留。

fdScript 显示文件名称的语系系统。通常£¨Finder（和标准文件软件包）以系统语系显示所有桌面对象的名称，它取决于系统在不同地区的配置。

fdScript 域字节中的高位的系统设定设定为 0，它使得Finder以当前系统语系显示文件名称。如果高位设定为 1，Finder（和标准文件软件包）就以记录在其它 7 位中的代码表示的语系显示文件和目录的名称。

fdXFlags 保留。

fdComment 在信息窗口中显示的备注的资源标识符编号，它是当用户选定一个文件并在文件菜单中选择“获取信息”命令时显示的。标识备注的编号是由 Finder 指派的。

fdPutAway如果用户把文件夹中的一个文件移至桌面上，它就是该文件夹的目录资源标识符编号。

目录信息记录

Finder操纵目录信息记录中的域，它们属于DInfo数据结构类型。您的应用程序不应该直接检查或设定所有这些域。

```
TYPE DInfo =
        RECORD
            frRect:        Rect;     {folder's window rectangle}
            frFlags:       Integer;  {flags}
            frLocation:    Point;{folder's location in window}
            frView:        Integer;  {folder's view}
        END;
```

域说明

frRect　当用户打开该文件夹时，Finder 显示的窗口矩形。

frFlags　保留。

frLocation 在父代窗口中该文件夹的位置。

frView　显示该文件夹的方式；这是由用户使用Finder的显示菜单中的命令设定的。

扩展的目录信息记录

Finder操纵扩展目录信息记录中的域，它们属于DXInfo数据结构类型。您的应用程序不应该直接检查或设定所有这些域。

```
TYPE    DXInfo =
        RECORD
            frScroll:      Point;    {scroll position}
            frOpenChain:   LongInt;  {directory ID chain of open}
                                       { folders}
            frScript:      SignedByte;{script flag and code}
            frXFlags:      SignedByte;{reserved}
            frComment:     Integer;    {comment ID}
            frPutAway:     LongInt;    {home directory ID}
        END;
```

域说明

frScroll 在 Finder 窗口中的滚动位置。Finder 并不需要在用户动作后立即存储这个位置。

frOpenChain 打开文件夹的目录资源标识符链。Finder 对目录资源标识符进行编号。Finder 并不需要在用户动作后立即存储这个信息。

frScript 显示文件夹名称的语系系统。通常，Finder 和标准文件软件包）以系统语系显示所有桌面对象的名称，它取决于系统在不同地区的配置。fd 语系域字节中的高位的系统设定为 0，它使得 Finder 以当前系统语系显示文件名称。如果高位设定为 1，Finder 和标准文件软件包）就以记录在其它 7 位中的代码表示的语系显示文件和目录的名称。但是，从 7. 1 版的系统软件开始，窗口管理器和对话管理器不同时支持多种语系，所以，总是使用系统语系在对话框、窗口标题和 Finder 所使用的其它用户界面元素中显示文件名称或目录名称。因此，在系统软件的语系能力完全实现之前，您应该把这个域保留。

frXFlags 保留。

frComment 在信息窗口中显示的备注的资源标识符编号，它是当用户选定一个文件夹并在文件菜单中选择“获取信息”命令时显示的。标识备注的编号是由 Finder 指派的。

frPutAway 如果用户把一个文件夹移至桌面上，它就是该文件夹从中移出的目录资源标识符编号。

例程

本节说明的是当您的应用程序不通过Finder操纵文稿时，它可以用来分辨替身文件的例程，以及需要确定与系统有关的文件夹的位置时，它可以使用的例程。

分辨替身文件

通常，当用户希望打开或打印文件时，您的应用程序并不需要关心它们是否是替身，因为Finder会在把它们传送给您的应用程序前，进行分辨。如果用户打开您的应用程序所创建的一个文稿的替身，Finder 向您的应用程序传送文稿本身，而不是替身的名称和位置。同样，当用户在您的应用程序中打开一个替身时，标准文件软件包向您的应用程序传送目标文稿的名称。）如果您的应用程序不通过 Finder 操纵文稿，它应该调用替身管理器函数 ResolveAliasFile 自己检查并分辨替身文件。为了完备起见，下面对 ResolveAliasFile 函数作一个说明。

ResolveAliasFile

如果您的应用程序不通过Finder操纵文稿，它应该使用ResolveAliasFile函数自己检查并分辨替身文件。

```
FUNCTION ResolveAliasFile (VAR theSpec: FSSpec;
                           resolveAliasChains: Boolean;
                           VAR targetIsFolder: Boolean;
                           VAR wasAliased: Boolean): OSErr;
```

theSpec　　您计划打开的一个文件或目录的文件系统规格说明记录。

resolveAliasChains
　　一个布尔值。如果您希望ResolveAliasFile分辨一个目录标识符链中的所有替身，直至它遇到了目标文件才停止，就把这个参数设定为TRUE。如果您只要ResolveAliasFile只分辨一个替身文件，并不需要找到目标文件，就把这个参数设定为FALSE.

targetIsFolder 返回的参数。如果参数 theSpec 中的文件系统规格说明记录指向一个目录或一个卷，ResolveAliasFile 在这个参数中返回TRUE；否则，返回FALSE。

wasAliased　　返回的参数。如果参数theSpec中的文件系统规格说明记录指向一个替身，ResolveAliasFile在这个参数中返回 TRUE；否则，返回FALSE。

说明

ResolveAliasFile函数在参数theSpec中返回您在开始时传送给theSpec的目标文件的名称和位置。

ResolveAliasFile函数首先在目录文件中检查参数theSpec中指定的文件或目录，以便确定它是一个替身还是一个文件或目录。如果对象不是替身，ResolveAliasFile不改变theSpec；对于目录或卷，它把targetIsFolder参数设定为TRUE，对于文件设定为FALSE；它把wasAlias设定为FALSE，并返回noErr。如果对象是一个替身，ResolveAliasFile先对它进行分辨，然后把目标放置在参数theSpec中，并把wasAlias标志设定为TRUE。

如果ResolveAliasFile 找到了指定的卷和父代目录，但是在此位置没有找到目标文件或目录，ResolveAliasFile 返回结果代码 fnfErr，并用说明目标的完全文件系统规格说明记录(即它的卷参考编号，父代目录标识符和文件名称或文件夹名称)填写参数 theSpec。尽管文件系统规格说明记录说明的对象不存在，它仍然是有效的。这个信息起“告诫”的作用，使您能够分辨失败，探索可能的解决方法。例如，您可以使用文件系统规格说明记录和文件管理器函数FSpCreate创建一个文件代替遗失的文件。

如果ResolveAliasFile 在分辨替身时收到一个错误代码，它把输入参数保持原样并退出，同时返回一个错误代码。除了上述这些结果代码外，ResolveAliasFile也能够返回任何资源管理器或文件管理器报告的错误。

特殊考虑

在调用ResolveAliasFile 函数前，您应该确保它是可用的，这可以用 Gestalt 函数和 gestaltAliasMgrAttr 选定程序来检查。

结果代码

noErr	0	无错误
nsvErr	-35	未找到卷
fnfErr	-43	目标未找到，但是找到了卷和父代目录，theSpec 参数包含一个有效的文件系统规格说明记录
dirNFErr	-120	父代目录未找到

亦请参阅

7-41 页的程序清单 7-13 说明了如何在应用程序自己的 MyOpen 函数中使用 ResolveAliasFile。文件系统规格说明记录在《Inside Macintosh 文件》*(Inside Macintosh : Files)* 一书中作了说明。替身和替身管理器和文件管理器的其它例程也在《Inside Macintosh 文件》*(Inside Macintosh : Files)* 一书中作了详细说明。《Inside Macintosh 操作系统实用程序》*(Inside Macintosh : Operating System Utilities)* 一书的“格式塔管理器”一章对 Gestalt 函数作了说明。

查找目录

为了获得您需要的那些与系统有关的目录的路径信息（请参阅7-41 页的“使用系统文件夹及和它有关的目录”)，您可以使用 FindFolder 函数。在这些目录中，您需要的可能只有预置选择、临时项目和废纸篓目录。例如，您可能会执行以下任务：检查用户的一个配置文件是否在预置选择文件中，在临时项目中创建一个临时文件，或者当您试图储存一个文件而应用程序存储盘区用完时，检查废纸篓目录的项目占用了多少存储，并向用户报告。

FindFolder

为了获得路径信息以便存取与系统有关的目录，应使用 FindFolder 函数。

```
FUNCTION FindFolder (vRefNum: Integer; folderType: OSType;
                     createFolder: Boolean;
                     VAR foundVRefNum: Integer;
                     VAR foundDirID: LongInt): OSErr;
```

vRefNum　　卷参考编号（对于启动磁盘是 kOnSystemDisk 常数），您希望在这个卷中确定一个目录。

folderType　　您要寻找的目录的一个 4 字符的文件夹类型，或者代表类型的一个常数。下面列出的是常数和它们所代表的 4 字符文件夹类型：

常数

```
kAppleMenuFolderType
                        = 'amnu';     {Apple Menu Items}
kControlPanelFolderType
                        = 'ctrl';     {Control Panels}
kDesktopFolderType= 'desk';     {Desktop Folder}
kExtensionFolderType= 'extn';   {Extensions}
kFontsFolderType   = 'font';     {Fonts folder}
kPreferencesFolderType = 'pref';{Preferences}
kPrintMonitorDocsFolderType
                        = 'prnt';     {PrintMonitor   }
                                      { Documents}
kStartupFolderType= 'strt';     {Startup Items}
kSystemFolderType  = 'macs';     {System Folder}
kTemporaryFolderType= 'temp';   {Temporary Items}
kTrashFolderType= 'trsh';  {single-user Trash}
kWhereToEmptyTrashFolderType
                   = 'empt';     {shared Trash on net}
```

createFolder　　如果一个目录不存在，在这个参数中传送 kCreateFolder 创建它，否则，就传送常数 kDontCreateFolder。

foundVRefNum FindFolder 返回的卷参考编号，该卷包含您在 folderType 参数中指定的目录。

foundDirID FindFolder 返回的目录 ID 编号，该目录是您在 folderType 数中指定的。

说明

对于在一个特定的卷（分别在 folderType 和 vRefNum 参数中指定）中的文件夹类型，FindFolder 在 foundVRefNum 参数中返回目录所在卷的参考编号，在 foundDirID 参数中返回目录的目录标识符。

用于给定卷的指定的文件夹在系统软件以后的版本中可能会在另一个卷中，因此不应该假定您在 vRefNum 中指定的卷和在 foundVRefNum 中返回的卷将是一样的。

您应该在 vRefNum 参数中指定一个卷的参考编号（对于启动磁盘指定常数 kOnS-ystemDisk）。

您应该在 folderType 参数中指定一个 4 字符文件夹类型，或者代表它的一个常数。应使用 kTrashFolderType 常数确定给定卷中，甚至一个文件服务器中当前用户的废纸篓目录。在一个文件服务器上，您可以使用 kWhereToEmptyTr-ashFolderType 常数确定所有登录用户的废纸篓子目录的父代目录的位置。

应在 createFolder 参数中使用常数 kCreateFolder，通知 FindFolder 创建一个尚未存在的目录；否则就使用 kDontCreateFolder 常数。只有当系统文件夹目录存在时才能创建系统文件夹中的目录。即使您在 createFolder 参数中指定 kCreateFo-lder 常数，FindFolder 也不会创建系统文件夹目录。

如果文件夹没有找到，FindFolder 函数就返回一个非零结果代码，它也能够返回文件管理器或存储器管理器报告的其它文件系统的错误。

特殊考虑

Finder 在位于系统文件中的一个 'fld#' 类型的资源中标识系统文件夹的子目录，及它们的文件夹类型。但是您不应该修改和依靠系统文件中 'fld#' 资源的内容；只应使用 FindFolder 函数查找适当的目录。

为了确定是否可以使用 FindFolder 函数，应使用 Gestalt 函数和 Gestalt 选定程序 gestaltFindFolderAttr。您在响应参数中测试 gestaltFindFolde-rPresent 常数所指示的标志位域。如果这个标志位被设定，那么 FindFolder 函数是存在的。

```
CONST        gestaltFindFolderPresent = 0;{if this bit is set, }
                                          { FindFolder is present}
```

结果代码

noErr	0	无错误
fnfErr	-43	在'fld#' 资源中未找到需要的类型，或者磁盘不支持系统文件夹，或者磁盘在卷标题中没有系统文件夹，或者磁盘没有支持桌面文件夹的桌面数据库，在后面 3 种情况下，都是没有找到文件夹
dupFNErr	-48	找到的是文件，而不是文件夹

亦请参阅

用 FindFolder 函数确定与系统有关的目录在 7-41 页的“使用系统文件夹及和它有关的目录”一节中作了说明。

资源

本节说明您为文件提供的资源，它们使 Finder 可以使用您的文件并把有关它们的信息转送给用户。这些资源是：

- 署名资源，它是用字符串（ 'STR' ）资源定义的，当用户双击您的应用程序创建的文稿时，Finder 用它标识并启动您的应用程序
- 一组资源（图标列表资源、小项目列表资源、大 4 位彩色图标资源、小 4 位彩色图标资源、大 8 位彩色图标资源和小 8 位彩色图标资源），它们代表了您的应用程序和它所创建的文稿的视觉形象，以及两个有关的资源：图标（ 'ICON' ）资源和彩色图标（ 'cicn' ）资源
- 文件参考资源，它把图标和它们所代表的文件类型链接起来，并允许用户通过把文稿图标拖拽至您的应用程序图标打开它
- 束资源，它把您的应用程序的署名资源、图标列表资源和文件参考资源组合在一起
- 一个未找到应用程序名字字符串，即一个字符串（ 'STR' ）资源，供应用程序的文稿使用，如果用户试图在您的应用程序不存在时，打开或打印它创建的一个文稿，就显示所谓应用程序的名称
- 应用程序文稿中的一个应用程序未找到消息字符串，即一个字符串（ 'STR' ）资源，它向用户解释为什么他不能打开或打印仅供您的应用程序使用的文稿
- 版本资源，使得用户能够容易地找到一个文件的版本，如果单个文件属于一个文件超集，用户能够找到整个文件超集的版本

有关使用 'SIZE' 资源支持固定样板文稿的信息，请参阅 7-34 页的“支持固定样板文稿”一节。

本节说明这些资源用Rez 编译程序编译后的结构。(Rez 资源编译程序可从 APDA 获得。)如果您有兴趣为这些资源创建Rez 输入文件，请参阅 7-6 页的“使用Finder 界面”中的详细信息。)

署名资源

每个创建文稿的应用程序都应该定义一个署名资源，当用户双击一个应用程序创建的文稿时，Fi nder 可以使用它标识并启动该应用程序。署名资源通常定义为一个字符串资源（即 ' STR' 类型的资源），它接受独有的 4 字符署名作为资源类型。例如，具有署名 WAVE 的一个应用程序使用一个字符串资源定义自己的署名资源为 'WAVE' 类型的资源。署名资源的资源标识符 应该是 0。

为了确保署名的唯一性，开发商必须在苹果电脑公司的 Macintosh 开发商技术支持部登记其应用程序的 4 字符署名。

本段说明定义为 'STR' 类型的署名资源经过 Rez 资源编译程序编译后的结构。署名资源的 Rez 输入文件格式和它编译后的输出形式不同。如果您只对创建一个署名资源有兴趣，请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件型”一节。

如果检验一个署名资源编译后的形式（请参阅图 7-16），您会发现它包含一个 Pascal 字符串，它指定了应用程序的名称、版本号和引用日期。

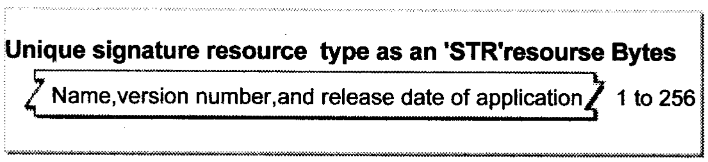


图 7-16 作为字符串（ ' STR' ）资源编译的署名资源的结构

若一个应用程序不通过版本资源提供专门的版本信息（请参阅 7-31 页的“提供版本资源”一节），当用户选定该应用程序，并在文件菜单中选择“获取信息”时，Fi nder 就显示储存在署名资源中的字符串。

一个图标列表资源被定义为两个 St ri ng [128] 类型的项目的阵列，第一个项目中的每个数位代表了 32 × 32 像素图标中的一个像素，第二个项目中的每位代表了 32 × 32 像素掩码中的一个像素。您可以使用一个高级工具，例如应用程序 ResEdi t（可以通过 APDA 获得），创建图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的图标列表资源转换成 Rez 输入。有关创建图标列表资源和其它对于用户代表文件的资源的附加信息，请参阅 7-11 页的“为 Fi nder 创建图标”一节。

图标列表资源

图标列表资源是几个图标资源之一，您创建它们是为了给用户提供应用程序或它创建的文稿类型直观的表示。图标列表资源的资源类型是 'I CN#'。所有的图标列表资源必须被标记为可清除的，它们的资源标识符必须大于 128。

当用户在显示菜单中选择“按照图标”显示时，如果用户使用的是黑白显示器或者您的应用程序未定义任何彩色图标资源，Fi nder 就显示在这个资源中定义的黑白图标；否则，Fi nder 就显示彩色图标。

一个图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-17 所代表的图标列表资源编译后的形式，您会发现它包含两个元素：

- 32 × 32 像素黑白图标。
- 32 × 32 像素全黑图标掩码，它给出了黑白图标及任何 32 × 32 像素彩色图标覆盖的面积。Finder 使用这个掩码把图标的轮廓线收到桌面的任何背景色或图案中。然后，Finder 把该资源中指定的黑白图标绘入这个外形。（或者把大 4 位彩色图标资源或大 8 位彩色图标资源中定义的彩色图标绘入。）

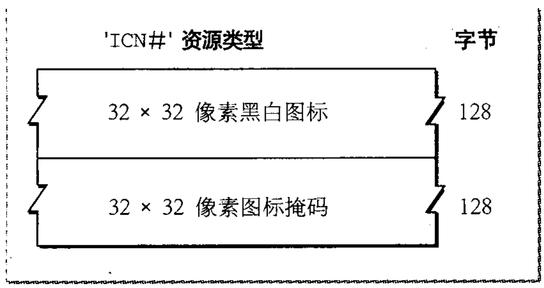


图 7-17 编译了的图标列表（ 'ICN#' ）资源的结构

为了创建图标列表资源中定义的 16 × 16 像素和彩色的图标版本（从而提供了整个图标库），您的应用程序还必须创建下列资源：小图标列表资源、大 4 位彩色图标资源、小 4 位彩色图标资源、大 8 位彩色图标资源和小 8 位彩色图标资源。它们编译后的格式在下几段中说明。创建它们的准则请参阅 7-11 页的“为 Finder 创建图标”一节。

当用户在显示菜单中选择“按照小图标”显示时，如果用户使用黑白显示器或者应用程序没有定义彩色图标资源，Finder就在窗口中显示这个资源中指定的小黑白图标，否则，就显示图标的彩色版本。同样，用户打开应用程序后，小黑白图标或它的彩色版本出现在应用程序菜单中，如果用户把应用程序或它的一个替身放在苹果菜单命令项目文件夹中，就出现在苹果菜单中。

小图标列表资源

小图标列表资源是您为一个图标库提供的几个资源之一。小图标列表资源的资源类型是 'ics#'。小图标列表资源必须被标记为可清除的，它必须和代表同样文件的图标列表资源有相同的资源标识符。

一个小图标列表资源被定义为两个String[32] 类型的项目的阵列，第一个项目中的每个数位代表了 16 × 16像素图标中的一个像素，第二个项目中的每个数位代表了 16 × 16 像素反白图标码中的一个像素。您可以使用一个高级工具，例如应用程序ResEdit创建小图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的小图标列表资源转换成 Rez 输入。有关创建小图标列表资源和其它对于用户代表文件的资源的附加信息，请参阅 7-11 页的“为Finder 创建图标”一节。

一个小图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-18 所代表的小图标列表资源编译后的形式，您会发现它包含下面两个元素:

- 在桌面环境中显示的 16 × 16 像素黑白图标
- 16 × 16像素全黑图标反白图标，它给出了该图标覆盖的面积。Finder 使用这个反白图标把图标的轮廓线收到桌面的任何背景色或图案中。然后，Finder 把该资源中指定的黑白图标绘入这个外形(或者把小 4 位彩色图标资源或小 8 位彩色图标资源中定义的彩色图标绘入。)

编译了的图标列表资源的格式请参阅 7-57 页；编译了的小 4 位彩色图标资源请参阅 7-60 页；编译了的小 8 位彩色图标资源请参阅 7-62 页。

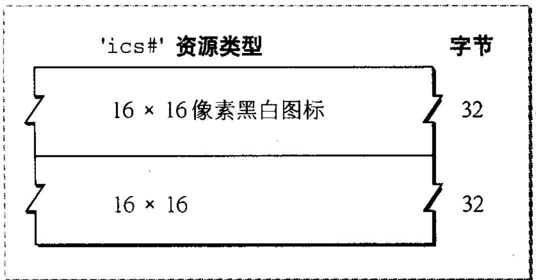


图 7-18 编译了的小图标列表（ 'ics#' ）资源的结构

当用户在显示菜单中选择“按照图标”显示时，如果用户的显示器的每个像素能够显示 4 位彩色数据，Finder 就在窗口中显示这个资源中指定的大 4 位彩色图标。同样，用户打开应用程序后，大 4 位彩色图标就出现在应用程序菜单中，如果用户把应用程序或它的一个替身放在苹果菜单命令项目文件夹中，就出现在苹果菜单中。

大 4 位彩色图标记录

大 4 位彩色图标列表资源是您为一个图标库提供的几个资源之一。大 4 位彩色图标列表资源的资源类型是 'icl4'。大 4 位彩色图标列表资源必须被标记为可清除的，它必须和代表同样文件的图标列表资源具有相同的资源标识符。

一个大 4 位彩色图标列表资源被定义为 String[512] 的类型；字符串中的每 4 位代表了 32 × 32 像素图标中的一个像素。您可以使用一个高级工具，例如应用程序 ResEdit，创建大 4 位彩色图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的大 4 位彩色图标列表资源转换成 Rez 输入。有关创建直观地代表文件的资源的信息，请参阅 7-11 页的“为 Finder 创建图标”一节。

一个大 4 位彩色图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-19 所代表的大 4 位彩色图标列表资源编译后的形式，您会发现它只包含 32 × 32 像素 4 位彩色图标供 Finder 显示。这个资源不为图标指定它的反白图标；相反，Finder 使用的反白图标是和这个资源具有同样资源标识符的图标列表资源所指定的。

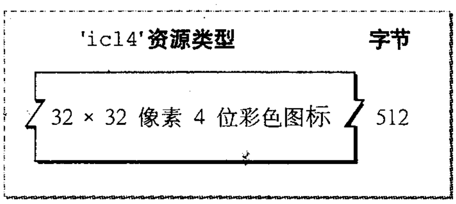


图 7-19 编译了的大 4 位彩色图标列表（ 'icl4' ）资源的结构

编译后的图标列表资源的格式请参阅 7-57 页。

小 4 位彩色图标记录

一个小 4 位彩色图标列表资源被定义为 String [128] 的类型；字符串中的每 4 个数位代表了 16 × 16 像素图标中的一个像素。您可以使用一个高级工具，例如应用程序 ResEdit，创建小 4 位彩色图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的小 4 位彩色图标列表资源转换成 Rez 输入。有关创建直观地代表文件的资源的信息，请参阅 7-11 的“为 Finder 创建图标”一节。

一个小 4 位彩色图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-20 所代表的小 4 位彩色图标列表资源编译后的形式，您会发现它只包含 16 × 16 像素 4 位彩色图标供 Finder 显示。这个资源不为图标指定它的反白图标；相反，Finder 使用的反白图标是和这个资源具有同样资源标识符的小图标列表资源所指定的。

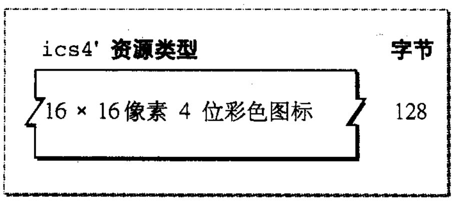


图 7-20 编译了的小 4 位彩色图标列表（'ics4'）资源的结构

编译后的图标列表资源的格式请参阅 7-57 页。编译后的小图标列表资源请参阅 7-58 页。

大 8 位彩色图标记录

大 8 位彩色图标列表资源是您为一个图标库提供的几个资源之一。大 8 位彩色图标列表资源的资源类型是 'icl8'。大 8 位彩色图标列表资源必须被标记为可清除的，它必须和代表同样文件的图标列表资源有同样的资源标识符。

当用户在显示菜单中选择“按照图标”显示时，如果用户的显示器的每个像素能够显示 8 位彩色数据，Finder 就在窗口中显示这个资源中指定的大 8 位彩色图标。同样，用户打开应用程序后，大 8 位彩色图标就出现在应用程序菜单中，如果用户把应用程序或它的一个替身放在苹果菜单命令项目文件夹中，就出现在苹果菜单中。

一个大 8 位彩色图标列表资源被定义为 String [1024] 的类型；字符串中的每个字节代表了 32 × 32 像素图标中的一个像素。您可以使用一个高级工具，例如应用程序 ResEdit，创建大 8 位彩色图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的大 8 位彩色图标列表资源转换成 Rez 输入。有关创建直观地代表文件的资源的信息，请参阅 7-11 页的“为 Finder 创建图标”一节。

一个大 8 位彩色图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-21 所代表的大 8 位彩色图标列表资源编译后的形式，您会发现它只包含 32 × 32 像素 8 位彩色图标供 Finder 显示。这个资源不为图标指定它的反白图标；相反，Finder 使用的反白图标是和这个资源具有同样资源标识符的图标列表资源所指定的。

编译后的图标列表资源的格式请参阅 7-57 页。

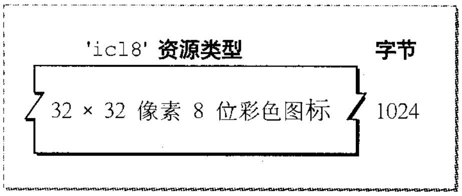


图 7-21 编译了的大 8 位彩色图标列表（ 'icl8' ）资源的结构

小 8 位彩色图标记录

小 8 位彩色图标列表资源是您为一个图标库提供的几个资源之一。小 8 位彩色图标列表资源的资源类型是 'ics8'。小 8
的，它必须和代表同样文件的图标列表资源有同样的资源标识符。

当用户在显示菜单中选择"按照小图标"显示时，如果用户的显示器的每个像素能够显示 8 位彩色数据，Finder 就在窗口中显示这个资源中指定的小 8 位彩色图标。同样，用户打开应用程序后，小 8 位彩色图标就出现在应用程序菜单中，如果用户把应用程序或它的一个替身放在苹果菜单命令项目文件夹中，就出现在苹果菜单中。

一个小 8 位彩色图标列表资源被定义为 String [256] 的类型；字符串中的每 8 个数位代表了 16 × 16 像素图标中的一个像素。您可以使用一个高级工具，例如应用程序 ResEdit，创建小 8 位彩色图标列表资源。然后，您可以在必要时使用 DeRez 反编译程序把您的小 8 位彩色图标列表资源转换成 Rez 输入。有关创建直观地代表文件的资源的信息，请参阅 7-11 页的"为 Finder 创建图标"一节。

一个小 8 位彩色图标列表资源定义一个图标，Finder 用它来显示它所代表的文件。如果您检验图 7-22 所代表的小 8 位彩色图标列表资源编译后的形式，您会发现它只包含 16 × 16 像素 8 位彩色图标供 Finder 显示。这个资源不为图标指定它的反白图标；相反，Finder 使用的反白图标是和这个资源具有同样资源标识符的小图标列表资源所指定的。

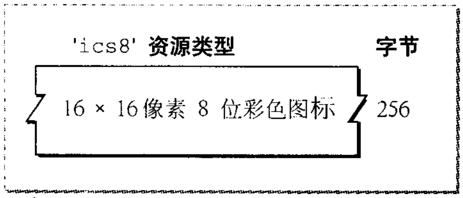


图 7-22　编译了的小 8 位彩色图标列表（ 'ics8' ）资源的结构

编译 h 区的图标列表资源的格式请参阅 7-57 页。编译后的小图标列表资源请参阅 7-58 页。

图标资源

当您希望在应用程序的某个元素（例如菜单、警告栏或对话框）中显示一个 32 × 32 像素的黑白图标时，就可以创建一个图标资源。图标资源的资源类型是 'ICON'。所有图标资源必须被标记为可清除的，它们的资源标识符必须大于 128。

您可以使用图标资源创建图标，它们类似于 Finder 在桌面环境中显示您的应用程序文稿所使用的图标；但是，和本节前面所介绍的资源类型不同，Finder 并不使用或显示任何您创建的'ICON'类型的资源。您的应用程序使用'ICON'类型的图标资源从您的应用程序出发显示图标。对于图标（ 'ICON' ）资源（您的应用程序为自已使用而创建的）、图标列表（'ICN#'）资源和前面所介绍的定义图标库所必须的其它资源（这是您的应用程序为 Finder 创建的），有时会感到迷惑，为了完备性，也为了减少可能造成的迷惑，这里也对图标资源进行说明。

有关创建图标列表资源和其它对于用户代表文件的资源的附加信息，请参阅 7-11 页的“为 Finder 创建图标”。

在一般情况下，您在菜单和对话框中使用图标资源，请参阅本书的“菜单管理器”和“对话管理器”两章。如果您提供彩色图标（'cicn'）资源，它和图标资源有同样的资源标识符，菜单管理器和对话管理器就对使用彩色显示器的用户显示彩色图标，而不是黑白图标。（例如，版面 2 的彩色警告栏对于它左上角的彩色图标指定了资源类型'cicn'。）

一个图标资源被定义为 String [128] 的类型；每个数位代表了 32 × 32 像素图标中的一个像素。正如图 7-23 所示，图标资源和指定反白图标的没有阵列的图标列表资源很相似。您可以使用一个高级工具，例如应用程序 ResEdit，创建图标资源。然后，您可以在必要时使用 DeRez 反编译程序把您的图标资源转换成 Rez 输入。

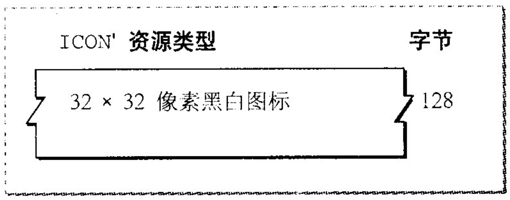


图 7-23 编译了的图标（'ICON'）资源的结构

彩色图标资源

当您希望在应用程序的某个元素（例如菜单、警告栏或对话框）中显示一个彩色图标时，就可以创建一个彩色图标资源。彩色图标资源的资源类型是 'cicn' 。所有的彩色图标资源必须被标记为可清除的，它们的资源标识符必须大于 128。

您可以使用彩色图标资源创建图标，它们类似于Finder在桌面环境中显示您的应用程序文稿所使用的图标；但是，Finder 并不使用或显示任何您创建的'cicn'类型的资源。您的应用程序使用'cicn'类型的图标资源从您的应用程序出发显示图标。对于彩色图标资源（即'cicn'类型的资源，它是您的应用程序为自己使用而创建的）和定义图标库所必须的大小 4 位及 8 位彩色图标列表资源（ 'ics4'、'icl4'、'ics8' 和 'icl8'类型的资源，它们是您为 Finder 使用而创建的），有时会感到迷惑，为了减少可能造成的迷惑，这里也对图标资源进行说明。

有关创建对于用户代表文件的图标库，包括彩色图标的附加信息，请参阅 7-11 页的“为 Finder 创建图标”。

在一般情况下，您在菜单、警告栏和对话框中使用彩色图标资源，请参阅本书的“菜单管理器”和“对话管理器”两章。如果您提供彩色图标（ 'cicn' ）资源，它和图标资源有同样的资源标识符（请参阅 7-63 页），菜单管理器和对话管理器就对使用彩色显示器的用户显示彩色图标，而不是黑白图标。您可以使用一个高级工具，例如应用程序 ResEdit，创建彩色图标资源。然后，您可以在必要时使用 DeRez 反编译程序把您的彩色图标资源转换成 Rez 输入。（例如，版面 2 的彩色警告栏对于它左上角的彩色图标指定了资源类型 'cicn'。）

有关彩色图标资源更详细的信息，请参阅《Inside Macintosh 绘图》*(Inside Macintosh: Imaging)* 一书。

文件参考资源

为了把图标和它们代表的文件类型进行链接，以及允许用户通过把文稿图标拖拽至您的应用程序图标打开您的应用程序，应对于您创建的每个图标列表资源创建一个文件参考资源。文件参考资源的资源类型是 'FREF'。所有文件参考资源的资源标识符必须大于 128，每个文件参考资源必须被标记为可清除的。

本段说明了Rez资源编译程序编译后的文件参考资源的结构。文件参考资源的 Rez 输入文件的格式和它编译后的输出形式是不同的。如果您只对创建文件参考资源有兴趣，请参阅 7-18 页的“创建文件参考资源”一节。

如果您检验图 7-24 所示的文件参考资源编译了的形式，您会发现它包含下列元素

- 文件类型。这是标识该资源代表的文件类型的 4 字符代码。有关文件类型请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节
- 局部标识符。Finder 使用这个编号把该资源中指定的文件类型映射到一个图标列表资源，后者在束资源中被指派了同样的局部标识符。7-57 页对图标列表资源作了说明；束资源在下一段中说明
- 空字符串。这个元素应该总是包含一个空的 Pascal 字符串

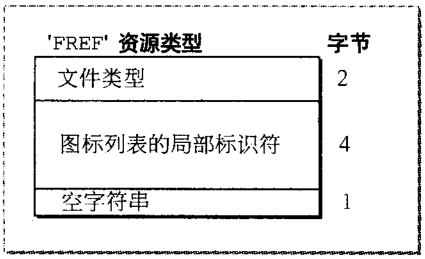


图 7-24 编译了的文件参考（ 'FREF' ）资源的结构

束资源

为了把您的应用程序的署名、图标列表资源和文件参考资源组合起来，应创建束资源。束资源的资源类型是'BNDL'。所有的束资源的资源标识符 都必须大于128，它们都必须是可清除的。

本段说明了Rez资源编译程序编译后的束资源的结构。束资源的Rez输入文件的格式和它编译后的输出形式是不同的。如果您只对创建束资源有兴趣，请参阅 7-20 页的“创建束资源”--节。

'BNDL' 资源类型	字节
应用程序署名	4
署名资源的资源标识符	2
阵列数（2）	2
局部标识符到图标列表资源标识符的映射	可变
对文件参考资源多余的局部标识符映射	可变

图 7-25 编译后的束（ 'BNDL' ）资源的结构

如果您检验图 7-25 所示的束资源编译了的形式，您会发现它包含下列元素：

- 应用程序署名。这是为 Finder 标识应用程序的独有的 4 字符代码（有关应用程序署名请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建程序和文件类型”一节）
- 署名资源的资源标识符。按照惯例，它应该设定为 0
- 阵列数。这个元素总应该包含 2
- 对于应用程序所提供的所有图标，其局部标识符到图标列表资源标识符的映射。请参阅图 7-26
- 对文件参考资源多余的局部标识符映射。这将在图 7-27 中说明

如果您检验把局部标识符映射到图标列表资源标识符的束资源编译后的部分，您会发现它包含下列元素：

- 资源类型。这个元素应该总是指定为‘ICN#’资源类型（即，一个图标列表资源）
- 应用程序提供的所有图标库的数目。这是该资源其余部分的局部标识符到图标列表资源标识符 的映射对的数目
- 图标列表资源的局部标识符。这个局部标识符必须和文件参考资源中指派给该图标列表资源的局部标识符匹配
- 在上一个元素中被指派了一个局部标识符的图标列表资源的资源标识符。为了直观地代表在文件参考资源（它包含上一个元素的局部标识符）中所说明类型的文件，Finder 使用该图标列表资源中说明的黑白图标和反白图标。Finder 还使用在具有同样资源标识符 的下列资源中所定义的图标：小图标列表资源、小 4 位彩色图标资源、小 8 位彩色图标资源、大 4 位彩色图标资源和大 8 位彩色图标资源

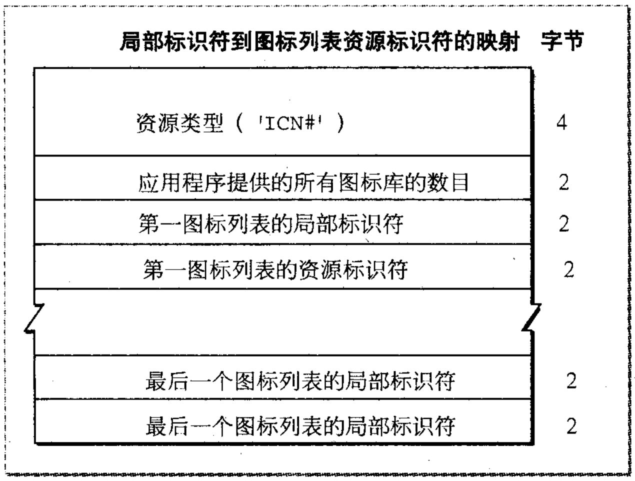


图 7-26 在一个束资源中把局部标识符映射到图标列表资源标识符

■ 对于代表应用程序文件类型的其余图标的局部标识符到图标列表资源标识符的映射对

图 7-27 给出了束资源的其余部分，它对文件参考资源标识符 指派了局部标识符。这个指派是多余的，因为 Finder 不把这些局部标识符映射到其它任何资源。这种标识符的指派在旧版 Macintosh 系统软件中需要进行，为了保持和以前的兼容性，现在仍然保留这一步。

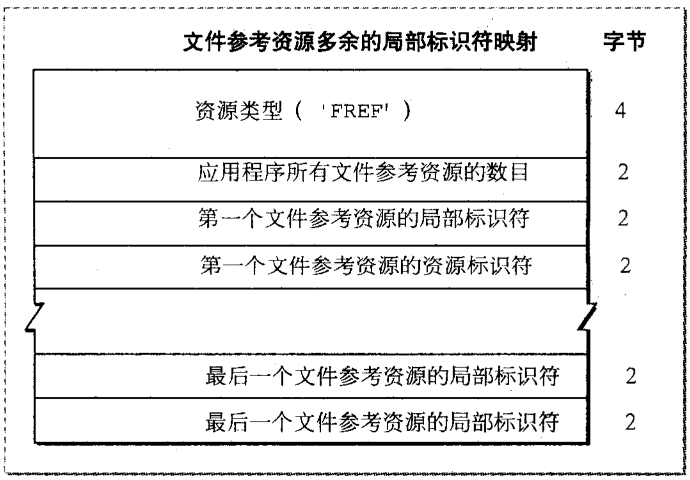


图 7-27 束资源中文件参考资源多余的局部标识符映射的结构

如果您检验剩余束资源编译了的部分，您会发现它包含下列元素:

- 资源类型。这个元素应该总是指定为‘FREF’资源类型。即，一个文件参考资源
- 代表应用程序文件类型的所有文件参考资源的数目。这是该资源其余部分的局部标识符到文件参考资源的映射对的数目
- 文件参考资源的局部标识符。这个局部标识符 可以是任何整数，只要和该束资源中所有其它文件参考资源的标识符都不相同即可
- 在上一域中指派了一个局部标识符的文件参考资源的资源标识符
- 其它文件参考资源的局部标识符到文件参考资源标识符的映射对，这些文件参考资源代表应用程序提供图标的文件类型

未找到应用程序名称字符串

当您的应用程序创建一个用户可以打开的文稿时，它应该在文稿的资源文件中包括一个未找到应用程序名字字符串。未找到应用程序名字字符串的资源类型是 'STR'，它的资源标识符 必须是 -16396，并且必须被标记为可清除的。这个字符串资源应该只包括您的应用程序的名称。有关把这个资源拷贝到您的文稿的资源分支中的附加信息，请参阅 7-27 页的“Finder 找不到您的应用程序时显示消息”一节。

如果您检验图 7-28 所示的一个编译了的未找到应用程序名字字符串，您会发现它完全由一个 Pascal 字符串组成，该字符串给出创建文稿的应用程序的名称。如果用户在应用程序不存在时试图打开或打印该应用程序创建的一个文稿，Finder就在一个警告栏中显示这个字符串。

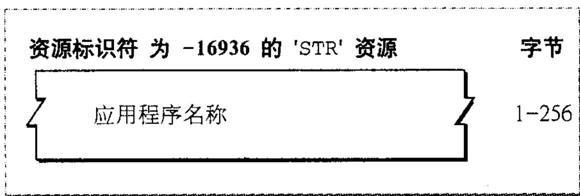


图 7-28 编译了的未找到应用程序名字字符串资源的结构

未找到应用程序消息字符串

当您的应用程序创建一个它自己使用但用户不能打开的文稿(例如预置选择文件)时，您的应用程序应该把文稿的创建者设定为一个登记的署名，这个署名必须和您的或其它的应用程序的署名都不同，同时您还应该在文稿的资源文件中包括一个应用程序未找到消息字符串。应用程序未找到消息字符串的资源类型是 'STR'，它的资源标识符必须为 -16397，并且必须被标记为可清除的。这个字符串资源应该包含一个消息，它解释为什么用户不能打开或打印这个文稿，请参阅 7-27 页的“Finder 找不到您的应用程序时显示消息”一节。

如果您检验图 7-29 所示的一个编译后的应用程序未找到消息字符串，您会发现它完全由一个Pascal字符串组成，该字符串解释为什么用户不能打开这个文稿。如果用户试图打开或打印--个文稿，而该文稿的创建者不是任何应用程序的署名，Finder 就在一个警告栏中显示这个字符串。(文件创建者和应用程序署名在7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节中作了说明。)



图 7-29 编译了的未找到应用程序信息字符串资源的结构

版本资源

您可以在任何文件中使用版本资源，使用户能够容易地找到文件的版本号。(如果这个文件是较大的文件集合的一部分，则找到文件超集的版本号。)版本资源的资源类型是‘vers’。资源标识符 编号为 1 的版本资源指定了单个文件的版本，资源标识符编号为 2 的版本资源指定了单个文件所属的文件超集的版本。

如果您的应用程序不包含资源标识符编号为 1 的版本资源，当用户在文件菜单中选择“获取信息”命令时，Finder 就在信息窗口中显示您的应用程序署名资源中的字符串。(请参阅 7-8 页的“给您的应用程序一个署名，给您的文稿一个创建者和文件类型”一节。)

本段说明了Rez资源编译程序编译后的版本资源的结构。版本资源的Rez输入文件的格式和它编译后的输出形式是不同的。如果您只对创建版本资源有兴趣，请参阅 7-31 页的“提供版本资源”一节。

如果您检验图 7-30 所示的版本资源编译后的形式，您会发现它包含下列元素:

- 以二进制编码的十进制格式表示的主要修正水平
- 以二进制编码的十进制格式表示的次要修正水平

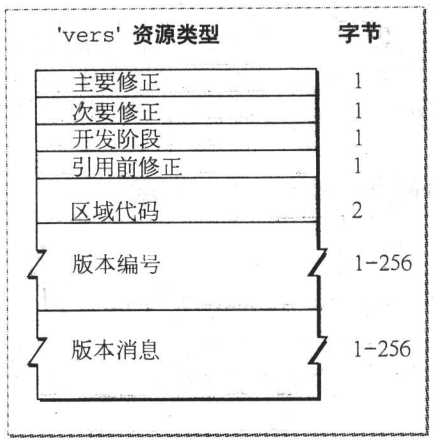


图 7-30 编译了的版本（'vers'）资源

- 开发阶段。可以在这个域中出现的值以及可以用来在Rez 输入文件中代表它们的常数如下:

值	常数	说明
0X20	development	alpha 预文件
0X40	alpha	alpha 文件
0X60	beta	beta 文件
0X80	release	正式引用文件

- 引用前修正水平。如果软件仍处于引用前的阶段，这个编号就指定了版本

- 区域代码。它标示这一版软件适用的语系系统。有关可以在这里指定的各种区域代码所代表的值的信息，请参阅《Inside Macintosh 文本》*(Inside Macintosh : Text)* 一书的“语系管理器”一章

- 版本编号。这个 Pascal 字符串标示软件的版本编号。如果用户打开显示控制板，单击显示版本框，然后在显示菜单中选择除了“按照图标”或“按照小图标”以外的任何命令，包含这个应用程序的 Finder 窗口就显示这个字符串

- 版本消息。这个 Pascal 字符串标示版本编号，以及文件的公司版权或者文件超集的产品名称。当用户选定这个文件并选择“获取信息”命令时，Finder 就在信息窗口中按照如下方式显示这个字符串:
  - 对于资源标识符为 1 的版本资源，这个字符串在信息窗口的版本域中显示。
  - 对于资源标识符为 2 的版本资源，这个字符串显示在信息窗口的顶部，文件名的下面，文件图标的旁边。

Finder 界面小结

Pascal 小结

常数

```
常数 {Gestalt selectors}
    gestaltFindFolderAttr  = 'fold';{selector for FindFolder}
    {interpreting Gestalt selector responses}
    gestaltFindFolderPresent= 0;{if this bit is set, }
                               { FindFolder is present}
    {for custom icons}
    kCustomIconResource    = -16455;{resource ID for }
                                    { custom icon}
    {for Finder flags}
    fHasBundle             = 8192;  {set if file has 'BNDL'}
    fInvisible             = 16384; {set if icon is invisible}
    kIsOnDesk              = $1;    {unused and reserved in }
                                    { System 7}
    kColor                 = $E;    {three bits of color coding}
    kIsShared              = $40;   {file can be executed by }
                                    { multiple users }
                                    { simultaneously}
    kHasBeenInited         = $100;  {file info is in desktop }
                                    { database}
    kHasCustomIcon         = $400;  {file or directory has a }
                                    { customized icon}
    kIsStationery          = $800;  {file is a stationery pad}
    kNameLocked            = $1000; {file or directory can't }
                                    { be renamed from Finder, }
                                    { and icon can't be changed}
    kHasBundle             = $2000; {file has a bundle resource}
    kIsInvisible           = $4000; {file or directory is }
                                    { invisible from Finder & }
                                    { from Standard File }
                                    { Package dialog boxes}
    kIsAlias               = $8000; {file is an alias file}
```

```
{for FindFolder}
kOnSystemDisk                = $8000; {use vRefNum for the }
                                      { boot disk}
kCreateFolder                = TRUE;  {create folder if it }
                                      { doesn't exist}
kDontCreateFolder            = FALSE; {don't create folder}

{for special folder types}
kSystemFolderType            = 'macs';    {System Folder}
kDesktopFolderType           = 'desk';    {Desktop Folder}
kTrashFolderType             = 'trsh';    {single-user Trash}
kWhereToEmptyTrashFolderType= 'empt';     {shared Trash on network}
kPrintMonitorDocsFolderType = 'prnt';     {PrintMonitor Documents}
kStartupFolderType           = 'strt';    {Startup Items}
kFontsFolderType             = 'font';    {Fonts}
kAppleMenuFolderType         = 'amnu';    {Apple Menu Items}
kControlPanelFolderType      = 'ctrl';    {Control Panels}
kExtensionFolderType         = 'extn';    {Extensions}
kPreferencesFolderType       = 'pref';    {Preferences}
kTemporaryFolderType         = 'temp';    {Temporary Items}
{alias types}
kContainerFolderAliasType    = 'fdrp';    {folder alias}
kContainerTrashAliasType     = 'trsh';    {Trash alias}
kContainerHardDiskAliasType= 'hdsk';      {hard disk alias}
kContainerFloppyAliasType    = 'flpy';    {floppy disk alias}
kContainerServerAliasType    = 'srvr';    {server alias}
kApplicationAliasType        = 'adrp';    {application alias}
kContainerAliasType          = 'drop';    {all other containers}
kSystemFolderAliasType       = 'fasy';    {System Folder alias}
kAppleMenuFolderAliasType    = 'faam';    {Apple Menu Items folder }
{ alias}
kStartupFolderAliasType      = 'fast';    {Startup Items folder alias}
kPrintMonitorDocsFolderAliasType = 'fapn';{PrintMonitor Documents }
                                          { folder alias}
kPreferencesFolderAliasType = 'fapf';     {Preferences folder alias}
kControlPanelFolderAliasType = 'fact';    {Control Panels folder alias}
kExtensionFolderAliasType    = 'faex';    {Extensions folder alias}
kExportedFolderAliasType     = 'faet';    {export folder alias}
kDropFolderAliasType         = 'fadr';    {drop folder alias}
kSharedFolderAliasType       = 'fash';    {shared folder alias}
kMountedFolderAliasType      = 'famn';    {mounted folder alias}
```

数据类型

```
TYPE    {Finder information records in the volume catalog file}
        FInfo =
        RECORD
            fdType:       OSType;        {file type}
            fdCreator:    OSType;        {filê creator}
            fdFlags:      Integer;       {Finder flags}
            fdLocation:   Point;         {file's location in window}
            fdFldr:       Integer;       {directory that contains file}
        END;

        FXInfo =
        RECORD
            fdIconID:     Integer;       {icon ID}
            fdUnused:     ARRAY[1..3]    OF Integer;
                                         {unused but reserved 6 bytes}
            fdScript:     SignedByte;    {script flag and code}
            fdXFlags:     SignedByte;    {reserved}
            fdComment:    Integer;       {comment ID}
            fdPutAway:    LongInt;       {home directory ID}
        END;

        DInfo =
        RECORD
            frRect:       Rect;          {folder's window rectangle}
            frFlags:      Integer;       {flags}
            frLocation:   Point;         {folder's location in window}
            frView:       Integer;       {folder's view}
        END;

        DXInfo =
        RECORD
            frScroll:     Point;         {scroll position}
            frOpenChain:  LongInt;       {dir ID chain of open folders}
            frScript:     SignedByte;    {script flag and code}
            frXFlags:     SignedByte;    {reserved}
            frComment:    Integer;       {comment ID}
            frPutAway:    LongInt;       {directory ID}
        END;
```

例程

分辨替身文件

```
FUNCTION ResolveAliasFile  (VAR theSpec: FSSpec;
                      resolveAliasChains: Boolean;
                      VAR targetIsFolder: Boolean;
                      VAR wasAliased: Boolean): OSErr;
```

查找目录

```
FUNCTION FindFolder    (vRefNum: Integer; folderType: OSType; createFolder:
                        Boolean;
                      VAR foundVRefNum: Integer;
                      VAR foundDirID: LongInt): OSErr;
```

C 小结

常数

```
enum {
    /*Gestalt selectors*/
    #define gestaltFindFolderAttr    'fold'  /*selector for FindFolder*/

    /*interpreting Gestalt selector responses*/
    gestaltFindFolderPresent        = 0      /*if this bit is set, */
                                             /* FindFolder is present*/
};
    /*for custom icons*/
    #define kCustomIconResource     -16455   /*resource ID for */
                                             /* custom icon*/

    /*Finder flags*/
    #define kIsOnDesk            0x1      /*unused and reserved in */
                                          /* System 7*/
    #define kColor               0xE      /*3 bits of color coding*/
    #define kIsShared            x40      /*file can be executed by */
                                          /* multiple users */
                                          /* simultaneously*/
    #define kHasBeenInited       0x100    /*file info is in desktop */
                                          /* database*/
    #define kHasCustomIcon       0x400    /*file or directory has a */
                                          /* customized icon*/
```

```
    #define kIsStationary           0x800    /*file is a stationery pad*/
    #define kNameLocked             0x1000   /*file or directory can't */
                                             /* be renamed from the */
                                             /* Finder, and icon can't */
                                             /* be changed*/
    #define kHasBundle              0x2000   /*file has a bundle */
                                             /* resource*/
    #define kIsInvisible            0x4000   /*file or directory is */
                                             /* invisible from Finder */
                                             /* & from Standard File */
                                             /* Package dialog boxes*/
    #define kIsAlias                0x8000   /*file is an alias file*/

enum {
    /*for Finder flags*/
    fHasBundle    = 8192,                    /*set if file has 'BNDL'*/
    fInvisible    = 16384                    /*set if icon is invisible*/
};
enum {
    /*for FindFolder*/
    kOnSystemDisk               = 0x8000     /*use vRefNum for the */
                                             /* boot disk*/
    #define kCreateFolder           true     /*create folder if it */
                                             /* doesn't exist*/
    #define kDontCreateFolder       false    /*don't create folder*/

    /*for special folder types*/
    #define kSystemFolderType       'macs'   /*System Folder*/
    #define kDesktopFolderType      'desk'   /*Desktop Folder*/
    #define kTrashFolderType        trsh'    /*single-user Trash*/
    #define kWhereToEmptyTrashFolderType
                                    'empt'   /*shared Trash*/
    #define kPrintMonitorDocsFolderType
                                    'prnt'   /*PrintMonitor Documents*/
    #define kStartupFolderType      strt'    /*Startup Items*/
    #define kFontsFolderType        'font'   /*Fonts*/
    #define kAppleMenuFolderType    'amnu'   /*Apple Menu Items*/
    #define kControlPanelFolderType 'ctrl'   /*Control Panels*/
    #define kExtensionFolderType    'extn'   /*Extensions*/
    #define kPreferencesFolderType  'pref'   /*Preferences*/
    #define kTemporaryFolderType    'temp'   /*Temporary Items*/
};
    /*for alias types*/
#define kContainerFolderAliasType   'fdrp'   /*folder alias*/
#define kContainerTrashAliasType    'trsh'   /*Trash alias*/
```

```
#define kContainerHardDiskAliasType     'hdsk'/*hard disk alias*/
#define kContainerFloppyAliasType       'flpy'/*floppy disk alias*/
#define kContainerServerAliasType       'srvr'/*server alias*/
#define kApplicationAliasType           'adrp'/*application alias*/
#define kContainerAliasType             'drop/*all other containers*/
#define kSystemFolderAliasType          'fasy/*System Folder alias*/
#define kAppleMenuFolderAliasType       'faam'/*Apple Menu Items
                                        folder *//* alias*/
#define kStartupFolderAliasType         'fast'/*Startup Items folder*/
                                        /* alias*/
#define kPrintMonitorDocsFolderAliasType'fapn'/*PrintMonitor Documents */
                                        /* folder alias*/
#define kPreferencesFolderAliasType     'fapf'/*Preferences folder
                                         alias*/
#define kControlPanelFolderAliasType    'fact'/*Control Panels fldr
                                         alias*/
#define kExtensionFolderAliasType       'faex'/*Extensions folder
                                         alias*/
#define kExportedFolderAliasType        'faet'/*export folder alias*/
#define kDropFolderAliasType            'fadr'/*drop folder alias*/
#define kSharedFolderAliasType          'fash'/*shared folder alias*/
#define kMountedFolderAliasType         'famn'/*mounted folder alias*/
```

数据类型

```
struct   FInfo {  /*Finder information records in the catalog file*/
         OSType            fdType;        /*file type*/
         OSType            fdCreator;     /*file creator*/
         unsigned short    fdFlags;       /*Finder flags*/
         Point             fdLocation;    /*file's location in
                                           window*/
         short             fdFldr;        /*directory that con-
                                          tains file*/
};

struct   FXInfo {
         short             fdIconID;      /*icon ID*/
         short             fdUnused[3];   /*unused but reserved
                                           6 bytes*/
         char              fdScript;      /*script flag and
                                           code*/
         char              fdXFlags;      /*reserved*/
         short             fdComment;     /*comment ID*/
         long              fdPutAway;     /*home directory ID*/
};
```

```
struct DInfo {
        Rect           frRect;       /*folder's window rectangle*/
        unsigned short frFlags;      /*flags*/
        Point          frLocation;   /*folder's location in window*/
        short          frView;       /*folder's view*/
};

struct DXInfo {
        Point          frScroll;     /*scroll position*/
        long           frOpenChain;  /*directory ID chain of open folders*/
        char           frScript;     /*script flag and code*/
        char           frXFlags;     /*reserved*/
        short          frComment;    /*comment ID*/
        long           frPutAway;    /*directory ID*/
};
```

例程

分辨替身文件

```
pascal OSErr ResolveAliasFile
                    (FSSpec *theSpec, Boolean resolveAliasChains,
                    Boolean *targetIsFolder, Boolean *wasAliased);
```

查找目录

```
pascal OSErr FindFolder (short vRefNum, OSType folderType,
                    Boolean createFolder, short *foundVRefNum,
                    long *foundDirID);
```

汇编语言小结

数据结构

FInfo 数据结构

0	fdType	长型	文件类型
4	fdCreator	长型	文件创建者
8	fdFlags	字	Finder 标志
10	fdLocation	长型	文件在窗口中的位置
14	fdFldr	字	包含文件的目录

FXInfo 数据结构

0	fdIconID	字	图标标识符
2	fdUnused	6 字节	保留
8	fdScript	1 字节	语系标志和代码
9	fdXFlags	1 字节	保留
10	fdComment	字	备注标识符
12	fdPutAway	长型	用户目录标识符

DInfo 数据结构

0	frRect	8 字节	文件夹的窗口矩形
8	frFlags	字	标志
10	frLocation	长型	文件夹在窗口中的位置
14	frView	字	文件夹的视图

DXInfo 数据结构

0	fr 语系	长型	滚动位置
4	frOpenChain	长型	打开的文件夹的目录标识符链
8	fs 语系	1 字节	语系标志和代码
9	frXFlags	1 字节	保留
10	frComment	字	备注标识符
12	frPutAway	长型	目录标识符

结果代码

noErr	0	无错误
nsvErr	-35	未找到卷
fnfErr	-43	对于 FindFolder: 在 'fld#' 资源中未找到需要的类型，或者磁盘不支持系统文件夹，或者磁盘在卷标题中没有系统文件夹，或者磁盘没有支持桌面文件夹的桌面数据库，在后面 3 种情况下，都是没有找到文件夹的数据库 对于 ResolveAliasFile: 目标未找到，但是找到了卷和父代 theSpec 参数包含一个有效的文件系统规范说明记录
dupFNErr	-48	找到的是文件，
dirNFErr	-120	父代目录未找到

词汇表

以汉语拼音为序

A

A5 区域　A5 world Finder″
按钮　button″

B

编目文件　catalog file″
变种代号　variation code″
标题条　title bar″
标准控制定义函数
standard control definition functions
标准状态　standard state″
部位代码　part code″

C

菜单　menu″
菜单标识符　menu ID″
菜单标题入口　menu title entry″
菜单定义过程　menu definition procedure″
菜单管理器　Menu Manager″
菜单记录　menu record″
菜单列表　menu list″
菜单条　menu bar″
菜单条定义函数　menu bar definition function″
菜单条入口　menu bar entry″
菜单条资源　menu bar resource″
菜单项目　menu item″
菜单项目入口　menu item entry″
菜单颜色入口记录　menu color entry record″
菜单颜色信息表　menu color information table″
菜单颜色信息表资源
menu color information table″
菜单资源　menu resource″
操作系统事件　operating-system event ″
操作系统事件队列
Operating System event queue″
操作系统事件管理器
Operating System Event Manager″
窗口　window″
窗口定义标识符　window definition ID″
窗口定义函数　window definition function″
窗口管理器端口　Window Manager port″
窗口记录　window record″
窗口类型　window type″
窗口列表　window list″
窗口区域　window region″
窗口颜色表　window color table″
窗口原点　window origin″
磁盘插入事件　disk-inserted event″
次切换　minor switch″

D

打印监控文稿文件夹
PrintMonitor Documents folder″
单选按钮　radio button″
当前菜单列表　current menu list″
当前进程　current process″
“当心”示警框　caution alert″
等效命令组合键　Command-key equivalent″
等效组合键　keyboard equivalent″
低级事件　low-level event″
定位名　location name″
定制示警框　costum alert box″
定制图标　costumized icon″
动作过程　action procedure″
端口矩形　port rectangle″
端口名称　port name″
对话管理器　Dialog Manager″
对话记录　dialog record″
对话框　dialog box″
对话颜色表资源　dialog color table resource″
对话资源　dialog resource″

F

发行物　edition″
非活动窗口　inactive window″
非活动控制部件　inactive control″
废纸篓文件夹　Trash folder″
分隔线　divider″
辅助窗口记录　auxiliary window record″
复选框　checkbox″

G

高级事件　high-level event″
高级事件队列　high-level event queue″

background process **后台进程**
不是当前与用户进行交互的进程。请与前台进程对照。

bundle bit **约束位**
文件的Finder 信息记录中的标志，用于告知Finder该文件有约束（'BNDL'）资源。文件的Finder 信息记录储存在卷编目文件中。Finder可以使用约束资源中的信息把图符和文件联系起来。

button **按钮**
一种显示在屏幕上的控制部件，形状为圆角矩形，中间有标题。当用户单击某个按钮时，应用程序就执行按钮标题给出的动作。按钮动作通常是在瞬间被完成。例如，执行对话框定义的操作指令和接收示警框中的错误信息。

catalog file **编目文件**
位于卷中的一个特殊文件，其中包含有关文件和文件夹在该卷中分级组织的信息。

caution alert **“当心”示警框**
一个示警框，用于警告用户：如果允许一个操作继续进行，将出现不希望的结果。“当心”示警框给用户提供了是继续该动作（单击“确认”按钮）还是停止该动作（单击“取消”按钮）的选择。“当心”示警框的标志是在示警框的左上角有一个带惊叹号的图符。参见“注意”示警框和“停止”示警框。

character code **字符代码**
代表一个特定字符的值。所生成的字符代码取决于虚拟键代码和修正键的状态。在罗马语系系统中，字符代码使用扩展 ASCII 码（美国信息交换标准代码）。

checkbox **复选框**
出现在屏幕上的一个控制，它是一个带有相应标题的小方框。复选框可显示两个状态之一：开（用方框内出现的X表示）和关。当用户单击复选框时，应用程序会切换它的状态。参见单选按钮。

close box **关闭框** 当前活动窗口标题栏左边的白色小方框。单击该框就可以关闭该窗口。

close region **关闭区域**
窗口的关闭格所占据的区域。亦参见关闭格。

Command-key equivalent **等效命令组合键**
专指用户请求的一种等效键盘命令：用户按住Command 键的同时按下另一个键（修正键除外）。

content region **内容区域**
窗口的一部分，可以显示文稿内容、调整大小框和窗口控制部件（包括滚动条）。

context **上下文**
进程管理器所维护的有关进程的信息。该信息包括进程的当前状态、其分区的地址和大小、类型、创建者、内存低端全局变量的一个拷贝、有关其'SIZE'资源的信息及进程序号。

control **控制部件**
一个用户可以通过鼠标进行操纵的屏幕上对象。通过操纵控制部件，用户可以立即执行动作或改变设定状态以修正以后的动作。

control color table **控制部件颜色表**
一个存储于项目颜色表资源中颜色说明，用于绘制控制的不同部位。

control definition function
控制定义函数
定义控制的外观和行为的一个函数。例如，一个控制定义函数可以绘制所定义的控制部件。参见标准控制定义函数。

control definition ID **控制定义标识符**
传送给创建控制例程的一个编号，用于指示控制部件的类型。它包含控制定义函数的资源标识符和一个变量代码。

control list **控制部件列表**
指向与窗口有关控制部件描述的一系列入口点。

Control Manager **控制管理器**
一个例程集合，应用程序用来创建和操纵控制，特别是窗口中控制。

Control Panels folder **控制板文件夹**
系统文件夹中用于储存控制板的目录。控制板允许用户改变 Macintosh 电脑的工作环境。

control record **控制记录**
ControlRecord 类型的一个数据结构，控制管理器用来储存对控制进行操作所需的信息。

current menu list **当前菜单列表**
一个数据结构，它包含当前菜单栏中所有菜单记录的句柄，以及应用程序插入列表中的所有子菜单或弹出式菜单的菜单记录。

current process **当前进程**
当前执行的进程，其 A5 区域是有效的。这个进程可以是后台进程，也可以是前台进程。

cursor **光标**
由一个 16 × 16 位方块定义的任意 256 位图象。鼠标驱动程序可以显示当前光标，并在用户移动鼠标时，把鼠标的移动映射到屏幕上的相对位置。

costum alert box **定制示警框**
一个示警框，其左上角是空白，或者所显示图标与“当心”、“停止”或“注意”示警框的图标完全不同。

costumized icon **定制图标**
用户或应用程序创建的一个图符，以资源标识符 -16455 储存在文件的资源分支中。如果一个文件带有定制图标，其 Finder 标志域中的 hasCustomIcon 标志位被设定。

data fork **数据分支**
文件的一部分，其中数据要用文件管理器来访问。这些数据常常对应于用户输入的数据；创建文件的应用程序可以用任何适当的方式储存和解释数据分支中的数据。

default button **缺省按钮**
在示警框或对话框中，用户按下回车键或Enter键时可以激发其动作的按钮。对话管理器自动在示警框的缺省按钮周围绘制粗轮廓线；而应用程序应当在对话框的缺省按钮周围绘制粗轮廓线。缺省按钮应该激发用户倾向做的动作，任何时候都应该是最安全的动作，即不造成数据丢失的动作。

desktop **桌面**
Macintosh电脑上显示的工作环境 屏幕上的灰色背景面积。

Desktop database **桌面数据库**
由 Finder 为所有容量超过 2MB 的卷而维护的数据库，其中包括图标、文件类型、应用程序、版本数据和说明。请和桌面文件比较。

Desktop file **桌面文件**
一个资源文件，Finder 用其为所用容量小于 2 MB 的卷储存图标、文件类型、应用程序、版本数据和说明。请和桌面数据库比较。

Desktop Folder **桌面文件夹**
位于各卷根层次的一个目录，用来供Finder储存出现在屏幕桌面上的图标的信息。用户看不见桌面文件夹，而只能在屏幕上看到所有安装卷的桌面文件夹内容的总和。

dial **刻度盘**
类似于滚动条的一个控制部件，图示用户可以设定的值的范围，或者简单地显示某物件的值、大小或位置，通常采用准模拟形式。

dialog box **对话框**
用于某些特殊或限定目的的窗口，例如：在执行用户命令之前，向用户征求信息。参见模式对话框、无模式对话框和可移动模式对话框。

dialog color table resource
对话颜色表资源
类型为‘dctb’的资源，可使应用程序采用系统缺省颜色以外的颜色显示对话框。

Dialog Manager **对话管理器**
应用程序用来实现警告栏和对话框的例程集合。

dialog record **对话记录**
类型为 DialogRecord 的数据结构，供对话管理器创建对话框和示警框之用。

dialog resource **对话资源**
类型为 'DLOG' 的资源，用于指定对话框的窗口类型、显示矩形和项目列表。

disabled item **停用项目**
在示警框或对话框中，对话管理器不向其报告用户事件的项目。如静态文字项目，它通常不对鼠标击键进行响应。

disk-inserted event **磁盘插入事件**
一个表明磁盘驱动器中已插入磁盘的事件。

display rectangle **显示矩形**
定义一个项目在示警框或对话框中的大小和位置的一个矩形。显示矩形可在一个项目列表中指定，并使用示警框或对话框中的局部坐标。

divider **分隔线**
菜单中使用的灰线，用于把菜单命令项目的各个组分隔开。

document window **文稿窗口**
用户输入文字、绘制图形或输入及操纵数据的一个窗口。

drag region **拖拽区域**
标题栏上除关闭框和缩放框以外的区域。用户可以通过拖拽该区域在桌面上移动一个窗口。

edition **发行物**
由发布者写入发行物容器中的数据。每当用户保存一个包含发布者的文稿时，发布者都把数据写入到发行物中，以便每次更新时其它文稿中的引用者都可以阅读这些数据。

enabled item **使能项目**
在示警框或对话框中，对话管理器向其报告用户事件的项目。例如：对话管理器可以报告对已使能 OK 按钮的鼠标击键。

event **事件**
事件管理器用于传送用户动作信息，改变应用程序的处理状态和其它从应用程序请求响应的手段。

event filter function **事件过滤函数** 一种应用程序定义的例程，可以补充对话管理器处理事件的能力。例如：事件过滤函数可以测试磁盘插入事件和允许后台应用程序接收更新事件。

Event Manager **事件管理器**
一个例程集，应用程序可以用其接收用户的动作信息，接收应用程序处理状态的改变通知以及与其它应用程序进行通信。

event mask **事件类型屏蔽参数**
一个整数，其中每一事件类型具有一个数位。您可指定事件类型屏蔽参数作为参数传送给事件管理器例程，从而指定您希望应用程序接收的事件类型。这样就停用（或“屏蔽”）了您不感兴趣的事件。

event record **事件记录**
具有EventRecord类型的数据结构，应用程序用以检索有关事件信息时。事件管理器在一个事件记录中返回事件类型的有关信息（例如：鼠标击键或键盘击键）和与该事件有关的信息。

Extention folder **功能扩展文件夹**
系统文件夹中的一个目录，用于储存系统功能扩展文件，例如：打印机和网络驱动程序以及具有 'INIT'、'scri' 和 'appe' 类型的文件。

file **文件**
储存在 Macintosh卷中的一个已命名的、有序的字节序列，它被分为一个数据分支和一个资源分支。

Finder
一个应用程序，与系统软件共同完成对文件的跟踪，并管理用户的桌面显示。

Fonts folder **字体文件夹**
一个位于系统文件夹中储存字体的目录。

forground process **前台进程**
当前与用户进行交互的进程；对于用户而言，是当前应用程序。前台进程会显示其菜单栏，其窗口在所有其它应用程序窗口的前面。请和后台进程比较。

frame **框架**
由窗口管理器自动绘制的窗口的一部分，包括标题条，关闭框和缩放框以及窗口轮廓线。

global coordinate system
全局坐标系统
代表 QuickDraw 所有潜在绘制空间的坐标系。全局坐标系统的原点（0，0）位于主屏幕的左上角。请与局部坐标系统相比较。

graphics port **图形端口**
一个完整、独立的绘制环境，带有独立的坐标系统。每个窗口都是在一个图形端口中绘制的。

gray area 灰色部位
滚动条中除了滚动箭头和滚动块以外的区域。当用户单击滚动条的灰色部位时，应用程序所移动的文稿显示面积的大小等于整个窗口面积减去一行（或列、横行、字符）。

gray region **灰色区域**
代表所有可见桌面部分的一个区域，即，代表显示器可见范围的圆角矩形。

grow image **增长映象**
用户使用尺寸框调整窗口大小时绘制在屏幕上的窗口新框架的轮廓线。

help balloon **旁述**
一个圆角矩形窗口，包含向用户进行解释的信息。旁述有一个尖角指针指向所注释的对象，看上去就像连环画中常用的包含对话的气球。旁述由用户在帮助菜单中启用。当旁述帮助打开时，只要用户把光标移至与旁述有关的区域，旁述就会出现。

hierarchical menu **嵌套式菜单**
带有子菜单的一个菜单。

high-level event **高级事件**
从一个应用程序发送到另一个应用程序的事件，请求传递信息或执行某个动作。

high-level event queue **高级事件队列**
事件管理器保维护的一个独立队列，用来储存发送给一个应用程序的高级事件。事件管理器为每个能够接收高级事件的已启动应用程序维护一个高级事件队列。

hot spot **热点**
鼠标驱动程序用来把光标和鼠标位置对齐的点。

icon **图标**
代表一个对象、概念或消息的图像。

icon family **图标组**
代表Finder所显示的对象（如应用程序或文稿）的一组图标。一个完整的图符组包括大图标（32 × 32 象素）和小图标（16 × 16象素），每一种都有相应的屏蔽掩码和三种不同的彩色：黑白、4 位彩色和素8 位彩色。

inactive control **非活动控制部件**
在当前上下文中没有意义或效果的一个控制部件，例如一个空窗口中的滚动杆。控制管理器用暗色显示非活动控制部件，或者用其它方法直观地显示其非活动状态。

inactive window **非活动窗口**
非用户当前工作窗口。

indicator **指示器**
在刻度盘或滑杆中的一个移动部分。用户通过移动指示器设定一个值，而应用程序通过移动指示器可以指定控制部件的当前设置。在滚动条中，滚动框是其指示器。

item color table resource **项目颜色表资源**
类型为 'ictb' 的资源，用来供应用程序使用与系统缺省字体及颜色不同的字形、风格、大小或颜色显示示警框或对话框的项目（为了使应用程序能够使用非标准的字形、风格或大小，用户必须拥有一个彩色显示器。）

item list **项目列表**
类型为 'DITL' 的资源，用于指定示警框或对话框中显示的项目，如按钮和静态文字。

item number **项目编号**
用于标识菜单或对话框中项目的一个整数。菜单项目按如下方式指定编号：第一个项目的编号为 1，第二个项目的编号为 2，依次排到菜单的最后一个项目。对话项目编号按照其在项目列表中的位置进行指定。例如，对话项目列表中第一项目的项目编号为 1。

keyboard equivalent **等效组合键**
等效组合键是一个或多个修正键和另一个键的键盘组合，当用户按下等效组合键时，就可调用相应的菜单命令。亦参见等效命令键。

key-down event **键按下事件**
一个事件，用于表明用户按下了键盘上的一个键。

key-up event **键释放事件**
一个事件，用于表明用户释放了键盘上的一个键。

local coordinate system **局部坐标系统**
图形端口的端口矩形所定义的坐标系。当窗口管理器创建一个窗口时，就把局部坐标系统的原点放置在窗口端口矩形的左上角。请比较全局坐标系统。

location name **定位名**
网络中 PPC 端口所在电脑位置的一个标识符。定位名称由一个对象字符串、一个类型字符串和一个区域组成。

low-level event **低级事件**
事件管理器返回的事件，用于报告低层次的硬件和软件所出现的事件。低级事件用于报告用户的动作和屏幕上窗口的改变，还可报告事件管理器没有其它事件可报告。请比较高级事件和操作系统事件。

major switch **主切换**
前台进程的改变。进程管理器把前台进程的内容和后台进程的内容（包括 A5 区域和低端存储器全局变量）进行交换，并把该后台进程带到前台，把原来的前台进程送到后台。参见上下文。

menu **菜单**
您可以在应用程序中使用的用户界面元素，允许用户从应用程序提供的选择和命令的列表中查看或选择一个项目。参见嵌套式菜单、弹出式菜单、下拉式菜单和子菜单。

menu bar **菜单条**
一个白色矩形，其高度足够用系统字体和系统字体大小显示菜单标题，其底部是一个象素宽的黑色边框。菜单条跨越启动屏幕的顶部，包含每个可用的下拉式菜单的标题。

menu bar definition function
菜单条定义函数
一个函数，用于绘制菜单条，当用户在菜单间移动光标时，执行和菜单显示有关的大多数绘制动作。该函数和菜单定义过程一起，定义了菜单的主要外观和行为。

menu bar entry **菜单条入口**
一个 mctID 和 mctItem 域都为 0 的菜单颜色入口记录。菜单条入口为应用程序的菜单条定义了颜色，并为菜单标题、菜单项目和背景定义了缺省颜色。

menu bar resource **菜单条资源**
类型为 'MBAR' 的资源，用于指定菜单条中每个菜单的次序和资源标识符。

menu color entry record
菜单颜色入口记录
类型为 MCEntry 的数据结构，用于定义应用程序的菜单条、菜单或菜单项目的颜色。颜色入口记录的前两个域 mctID 和 mctItem 定义了该入口是一个菜单条入口、菜单标题入口，还是菜单项目入口。

menu color information table
菜单颜色信息表
菜单管理器所维护的菜单颜色入口记录的阵列，用于为所显示菜单的菜单条、菜单标题、菜单项目的文字和特征以及背景定义标准颜色。如果不在该表上添加任何入口，菜单管理器会使用缺省的白底黑色绘制菜单。

menu color information table
菜单颜色信息表资源
类型为 'mctb' 的资源，用于为应用程序的菜单条、菜单和菜单项目指定颜色。

menu definition procedure
菜单定义过程
在指定菜单中绘制所有菜单项目的一个过程。该过程和菜单条定义函数一起，定义了菜单的主要外观和行为。

menu ID **菜单标识符**
为应用程序中的一个菜单所指派的编号。应用程序中的每个菜单必须有自己唯一的菜单标识符。

menu item **菜单项目**
菜单中的一个矩形，其文字和其它特征标识了一个用户可以选择的命令。

menu item entry **菜单项目入口**
一个菜单颜色入口记录，其 mctID 和 mctItem 域都包含非零值。菜单项目入口为指定菜单中的项目的标记、文字和等效命令组合键定义了颜色。它还定义了菜单的缺省背景色。

menu list **菜单列表**
一个数据结构，其中包含一个或多个菜单记录的句柄（一个菜单列表也可能是空的）。请比较当前

菜单列表。

Menu Manager **菜单管理器**
应用程序用来创建、显示和管理其菜单的例程的集合。

menu record **菜单记录**
类型为 MenuInfo 的数据结构，用于菜单管理器维护有关菜单的信息。

menu resource **菜单资源**
类型为 'MENU' 的资源，用于指定菜单标题和菜单中项目的个别特征。

menu title entry **菜单标题入口**
一个菜单颜色入口记录，其mctID 包含非零值，mctItem域为 0。菜单标题入口定义了特定菜单的标题和项目的颜色，以及背景色。它还定义了菜单条缺省颜色。

minimum partition size **最小分区**
应用程序分区大小的下限，低于该值，应用程序将不能运行。

minor switch **次切换**
进程上下文的改变的改变。进程管理器交换一个进程的内容使后台进程拥有处理器时间，但是不把后台进程带到前台。

modal dialog box **模式对话框**
一个对话框，它使用户处于一种只能在该对话框中工作的状态或"模式"。模式对话框和示警框相似。用户不能移动模式对话框，只能通过单击其按钮将其取消。参见无模式对话框和可移动模式对话框。

modaless dialog box **无模式对话框**
一个对话框，看上去像一个没有调整框或滚动条的文稿窗口。用户可以移动一个无模式对话框，将其激活或停用，并可像其它文稿窗口一样将其关闭。参见模式对话框和可移动模式对话框。

modifier keys **修正键**
Shift 键、Option 键、Command 键、Control 键和 Caps Lock 键。

mouse-down event **鼠标按下事件**
一个事件，表明用户按下了鼠标按钮。

mousemoved event **鼠标移动事件**
一个事件，表明光标在某指定区域之外。

mouse-up event **鼠标释放事件**
一个事件，表明用户释放了鼠标按钮。

movable modal dialog box
可移动模式对话框
带有标题栏（没有关闭框）的一个模式对话框，用户可以通过标题栏拖拽该对话框。参见对话框、模式对话框和无模式对话框。

note alert **"注意"示警框**
一个示警框，用于向用户通报一个小的错误，这种错误继续存在不会造成任何灾难。通常，一个"注意"示警框只是提供信息，用户只需单击"确认"按钮。"注意"示警框的标志是在示警框的左上角有一个带面孔的图标及类似卡通的对话气球。参见"当心"示警框和"停止"示警框。

null event **空事件**
一个事件，用于表明在应用程序的事件流中没有所要求类型的事件。

offset point **偏移点**
区域中的一个点，其相对于该区域包容矩形左上角的水平和垂直偏移量和一个特定点的偏移量相同。DragGrayRgn 使用一个偏移点限制一个区域的移动，并计算移动的距离。

operating-system event **操作系统事件**
由事件管理器返回的事件，用于传送有关应用程序操作状态变化的信息（挂起和恢复事件）和报告用户已将鼠标移出应用程序指定的区域（鼠标移动事件）。请比较低级事件和高级事件。

Operating System Event Manager
操作系统事件管理器
一个低级例程集，可以管理操作系统事件队列。

Operating System event queue
操作系统事件队列
操作系统事件管理器创建和维护的队列。操作

系统事件管理器可以检测和报告与硬件相关的低级事件（如：鼠标击键事件、按键事件和磁盘插入事件），并将这些事件放入操作系统事件队列之中。

part code **部位代码**
从 1 到 253 之间的一个整数，用于代表控制中的一个特定部分。当用户按下鼠标按钮时，FindControl 和 TrackControl 函数会返回表示光标位置的一个部位代码。

pop-up menu **弹出式菜单**
在菜单栏以外的地方出现的一个菜单。在实现弹出式菜单时，控制管理器提供了一个控制定义函数供应用程序使用。

port name **端口名称**
电脑上特定应用程序的唯一标识符，供应用程序间通讯使用。端口名称由一个名称字符串、一个类型字符串和一个语系代码组成。

port rectangle **端口矩形**
图形端口数据结构中的一个入口，在《Inside Macintosh：绘图》一书中作了说明。通常，端口矩形代表了绘图可用的图形端口区域，即一个窗口的内容区域。

Preference folder **预置文件夹**
一种定位在系统文件夹中、用于存储文件的目录，这些文件可以记录用户为应用程序在特定 Macintosh 电脑上的配置设定状态。

preferred partition size **推荐分区**
应用程序能够最有效运行的分区大小。操作系统在启动应用程序时尽力保证该分区大小。

PrintMonitor Documents folder
打印监控文稿文件夹
位于系统文件夹中的一个目录，用于储存等待打印的假脱机文稿。

process **进程**
一个启动的应用程序，或者在某些情况下是一个启动的桌面附件。（只有不是在另一个应用程序的上下文中启动的桌面附件才能被认为是进程。）

process serial number **进程序号**
进程管理器指派的一个编号，用来标识本地电脑单次启动时的一个应用程序的特定实例。

pull-down menu **下拉式菜单**
在菜单条中由菜单标题（一个词或一个图标）标识的菜单称为下拉式菜单。

query document **询问文稿**
文件类型为 'qery' 的一个文件，它以适当的数据库或数据源的格式包含命令和数据。应用程序使用高级数据存取管理器例程来开启询问文稿。

radio button **单选按钮**
屏幕上以小圆圈形式出现的一个控制部件。单选按钮显示下面两种设定状态之一：开（用圆圈中的黑点表示）或关。一个选单按钮总是一组相关单选按钮中的一个，在该组单选按钮中同时只能有一个处于开的状态。当用户单击一个无标记的单选按钮时，应用程序将这个单选按钮打开，同时把同组按钮中的另一个关闭。

Rescued Items from volume name folder
来自卷名的拯救项目文件夹
位于废纸篓目录中的一个目录，如果Finder在临时项目文件夹中找到一个文件（通常发生在系统崩溃时），就在系统启动、重新启动或关机时创建该目录。来自卷名的拯救项目文件夹按照临时项目文件夹所在的卷命名。当用户清空废纸篓时，所有的拯救项目文件夹都将消失。

resource **资源**
按照定义的结构储存在文件的资源分支中的任何数据。资源中的数据根据其资源类型进行解释。

resource fork **资源分支**
文件中包含文件资源的部分。资源分支由资源映象和资源组成。

resource ID **资源标识**
标识一个给定资源类型的指定资源的编号。

resource type **资源类型**
用于唯一地标识一种指定类型资源的 4 字符序列。

resume event **恢复事件**
一个事件，用于表明一个应用程序被切换回到前台，并可以重新和用户发生交互。

return receipt 回执
一个高级事件，用于表明您的应用程序发送给其它应用程序的高级事件是否被接收了。

scroll arrow 滚动箭头
滚动条两端的箭头。当用户单击一个滚动箭头时，应用程序按照箭头的方向把文稿或列表移动一行（或某些类似的度量）。如果用户按住鼠标按钮，同时光标在滚动箭头上，应用程序按照箭头的方向连续地移动文稿或列表。

scroll bar 滚动条
一个控制部件，用于改变文稿在窗口中的显示部分。滚动条是一个淡灰色矩形，滚动箭头在它的两端。一个窗口可以有一个水平滚动条或一个垂直滚动条，或者两者都有。垂直滚动条在窗口的右边，水平滚动条在窗口底部。滚动条内有一个被称为滚动框的矩形。滚动条的其余部分称为灰度部位。用户可以通过操纵滚动条的各个控制部分移动文稿。

scroll box 滚动框
在滚动杆中上下或左右滑动的一个方块。滚动框在滚动条中的位置表示了窗口内容相对于整个文稿的位置。当用户拖拽滚动框时，应用程序就会显示文稿的不同部分。

signature 签名
一个资源，其类型用一个4 字符序列定义，该资源使Finder 能够唯一地标识应用程序。签名位于应用程序的资源分支中。

size box 调整框
一个方块，位于可改变大小窗口的右下角。拖拽调整格可以调整窗口的大小。

size region 调整区域
窗口的调整框所占据的区域。参见调整框。

size resource 规模资源
类型为 'SIZE' 的资源，用于指定一个应用程序的运行特征、最小分区和推荐分区。

slider 滑杆
一个控制部件，就像滚动条，用于图示用户可以设定的值的范围，或者简单地显示某物件的值、大小和位置，通常采用准模拟的形式。

standard control definition functions 标准控制定义函数
在系统文件中作为资源 'CDEF' 储存的 3 个控制定义函数。资源标识为 0 的'CDEF' 资源定义了按钮、复选框和单选按钮的外观和行为；资源标识符为 1 的'CDEF' 资源定义了滚动条的外观和行为；资源标识符为 63 的 'CDEF' 资源定义了弹出式菜单的外观和行为。

standard state 标准状态
应用程序认为对窗口最佳的尺寸和位置。

Startup Items folder 启动项目文件夹
位于系统文件夹中的一个目录，用于储存用户希望在 Finder启动时需要启动的应用程序和桌面附件。

stationary pad 样板
用户创建的一个文稿，用作其它文稿的样板。Finder 通过设定该文件文件信息记录中 Finder 标志域的 isStationery 位，把文稿标识为样板。如果要求一个应用程序打开一个样板，它应该把样板的内容拷贝到一个新的文稿中，然后在一个无标题文件窗口中打开新文稿。

stop alert “停止”示警框
一个示警框，用于通知用户：发生的问题非常严重，用户的动作无法完成。“停止”示警框通常只有一个按钮（“确认”），因为用户只能接受该动作无法完成的事实。“停止”示警框的标志是在示警框的左上角有一个示意停止的举起的手的图标。参见“当心”示警框和“注意”示警框。

structure region 结构区域
窗口所占据的整个屏幕区域，包括窗口框架和内容区域。

submenu 子菜单
附属于另一个菜单的菜单。

suspend event 挂起事件
当主切换或次切换发生时，指示将暂停执行您应用程序下次调用 WaitNextEvent 或 EventAvail 时，将被挂起。参见恢复事件。

system alert sound 系统示警声
储存在系统文件中的一个声音资源，当系统软

件或应用程序使用声音管理器过程 SysBeep 时进行播放。用户可以使用声音控制板选定所播放的语音。

System file **系统文件**
位于系统文件夹中的一个文件，其中包含基本系统软件和某些系统资源，例如声音和键盘资源。系统文件在下述方面像一个文件夹：尽管它外观像一个手提箱式图符，但连击它可开启窗口，展示储存在系统文件中的可移动资源文件（例如声音、键盘布局和语系系统资源集合）。

System folder **系统文件夹**
包含用来启动 Macintosh 电脑的软件的一个目录。系统文件夹包含一组文件夹，分别储存有关的文件，例如应用程序在启动时可能需要的预置选择文件。

Temporary Folder **临时项目文件夹**
位于卷的根层次的一个目录，用于储存应用程序 用户看不到临时项目文件夹。

text style specification **文本风格表** 在项目颜色表资源中，对于可编辑文本项或静态文本项的字形、风格、大小和颜色的规格说明。

title bar **标题条**
位于窗口顶部的条框，用于显示窗口的名称，包含关闭格和缩放格，并通过栏中跑道线的状态表明该窗口是否当前活动窗口。

Toolbox Event Manager **工具箱事件管理器**

参见事件管理器。

Trash folder **废纸篓文件夹**
卷的根层次的一个目录，用于储存用户移至废纸篓图标的文件。在打开废纸篓图标后，用户会看到已经移至废纸篓图标的所有项目的集合，即，来自所有安装卷的相应废纸篓目录的和。在网络环境中设置成用户间共享文件的Macintosh电脑为远程用户在其共享废纸篓目录中保持独立的废纸篓子目录。仅当该目录的用户选择"清空废纸篓"命令时，Finder 才清空一个废纸篓目录（对于文件服务器，是废纸篓子目录）。

update event **更新事件**
一个事件，用于表明一个窗口的内容需要更新。

update region **更新区域**
窗口管理器所维护的区域，其中包括窗口内容区中需要更新的一部分。事件管理器在必要时会根据更新区域的内容生成更新事件，报告应用程序更新窗口。

user state **用户状态**
用户为窗口设定的尺寸和位置。

variation code **变种代号**
一个数字，用于在单个窗口定义函数或控制定义函数所支持的变种中选定一个。变种代号储存在窗口定义标识符或控制定义标识符的低 4 位。参见控制定义函数、控制定义标识符、窗口定义函数和窗口定义标识符。

virtual key code **虚拟键代码**
代表用户按下或释放的键的一个值。这个值对于键盘上一个特定的物理键总是同样的。请比较字符代码。

visible region **可见区域**
窗口图形端口在屏幕上可以看到的部分，即没有被其它窗口覆盖的部分。

window **窗口**
屏幕上显示信息的一个区域，这些信息包括用户文稿以及示警框和对话框中的内容。用户可以打开或关闭一个窗口，在桌面环境中移动，有时还可以改变大小、滚动，或者编辑其中的文字内容。

window color table **窗口颜色表**
一个数据结构，窗口管理器在其中储存绘制窗口框架及高亮度显示选定的文字所使用的颜色。

window definition function **窗口定义函数**
定义一个窗口的主要外观和行为的一个函数。窗口管理器调用窗口定义函数绘制窗口框架、确定光标所在的窗口区域、绘制窗口的调整框和缩放框、移动窗口并调整窗口的大小，以及计算窗口的结构和内容区域。

window definition ID **窗口定义标识符**
一个整数，它的高端 12 位指定了窗口定义函数的资源标识符，低端 4 位指定了一个可选的变种代号。当创建一个新窗口时，您的应用程序可以在 'WIND' 资源中把资源标识符作为一个域，也可以把它作为传送给 NewWindow 或NewC-

Window 的一个参数。

window list **窗口列表**
窗口管理器维护的关于桌面上所有窗口的一个列表。最前面的窗口是窗口列表中的第一个。其余窗口按照它们在桌面上叠放的次序排列。

Window Manager port **窗口管理器端口**
一个图形端口，它代表主显示器上的桌面区域，即一个圆角矩形，它占据了主显示器上除了标题栏以外的所有区域。

window origin **窗口原点**
窗口的左上角。通常它的坐标被指定为(0, 0)，窗口原点是以窗口的局部坐标表示的。

window record **窗口记录**
类型为 WindowRecord (或 CWindowRecord)的一个数据结构，窗口管理器在其中储存一个窗口的特征，包括窗口的图形端口、标题、可见性状态和控制列表。

window region **窗口区域**
窗口的专门用途区域。参见关闭区域、内容区域、拖拽区域和缩放区域。

window type **窗口类型**
说明一个窗口的特征的集合，例如窗口框架的形状和它的标题栏的功能。

zoom box **缩放框**
窗口标题栏右边的一个方块，用户可以通过它在窗口的两种大小（用户状态和标准状态）之间更换。

zoom region **缩放区域**
窗口的缩放框占据的区域。参见缩放框。

索引

以英文字母为序

符号

‘ **** ’ 文件类型 7-20
‘????’ 文件类型 7-20

数字

16X16 像素（小）图标 16-by-16 pixel (small) icons 7-11，7-15
16X16 像素（小）图标 16-by-16 pixel (small) icons 7-11，7-15
32X32 像素（大）图标 32-by-32 pixel (large) icons 7-11，7-13
32X32 像素（大）图标 32-by-32 pixel (large) icons 7-11，7-13
4 位彩色图标 4-bit color icons 7-11
4 位彩色图标 4-bit color icons 7-11
8 位彩色图标 8-bit color icons 7-11
8 位彩色图标 8-bit color icons 7-11

A

A/UX 和取消事件 A/UX and cancel events 2-46
A5 区域 A5 world
  上下文切换 and context switching 2-16
  大小 size of 2-32
ACount 全局变量 ACount global variable 6-172
AEIMP（苹果事件进程间通信协议） AEIMP (Apple Event Interprocess Messaging Protocol) 2-13，2-67
AEProcessAppleEvent 函数 AEProcess-AppleEvent function 2-68，2-78
AESend 函数 AESend function 2-73，2-78，2-100
ANumber 全局变量 ANumber global variable 6-172
AcceptHighLevelEvent 函数 AcceptHigh-LevelEvent function 2-69，2-70 至 2-71，2-87，2-90 至 2-92
AddResMenu 过程 AddResMenu procedure。参见 AppendResMenu 过程
Alert 函数 Alert function 6-106 至 6-108
AppendDITL 过程 AppendDITL procedure 6-54，6-125 至 6-127
AppendMenu 过程 AppendMenu procedure 3-64 至 3-67，3-124 至 3-125
AppendResMenu 过程 AppendResMenu procedure 3-20，3-68 至 3-70，3-128 至 3-129
AtMenuBottom 全局变量 AtMenuBottom global variable 3-151
AuxCtlRec 数据类型 AuxCtlRec data type 5-76 至 5-77，5-107
AuxWinRec 数据类型 AuxWinRec data type 4-73 至 4-74

B

BeginUpdate 过程 BeginUpdate procedure 2-47，4-50，4-106 至 4-107
BringToFront 过程 BringToFront procedure 4-90
Button 函数 Button function 2-108

C

CWindowPeek 数据类型 CWindowPeek data type 4-65
CWindowPtr 数据类型 CWindowPtr data type 4-65
CWindowRecord 数据类型 CWindowRecord data type 4-65 至 4-68
CalcMenuSize 过程 CalcMenuSize procedure 3-143，5-28
CalcVis 过程 CalcVis procedure 4-119
CalcVisBehind 过程 CalcVisBehind procedure 4-119
Caps Lock 键 Caps Lock key 2-20
CautionAlert 函数 CautionAlert function 6-111
CheckItem 过程 CheckItem procedure 3-61 至 3-62，3-136
CheckUpdate 函数 CheckUpdate function 4-116
ClearMenuBar 过程 ClearMenuBar procedure 3-110
ClipAbove 过程 ClipAbove procedure 4-116 至 4-117
CloseDialog 过程 CloseDialog procedure 6-119 至 6-120
CloseWindow 过程 CloseWindow procedure 4-60，4-104 至 4-105
ColorReserved 标志 colorReserved flag 7-48

ColorSpec 数据类型 ColorSpec data type 4-72，5-78 至 5-79
Command 键 Command key 2-20
Command-Shift-数字键序列 Command-Shift-number key sequences 2-39，2-86
Control 键 Control key 2-20
ControlRecord 数据类型 ControlRecord data type 5-73 至 5-75
CountDITL 函数 CountDITL function 6-128
CountMItem 函数 CountMItems function 3-81，3-141
CtlCTab 数据类型 CtlCTab data type 5-77 至 5-80

D

DABeeper 全局变量 DABeeper global variable 6-172
DAStrings 全局变量 DAStrings global variable 6-172
DIBadMount 函数 DIBadMount function 2-56
DITLMethod 数据类型 DITLMethod data type 6-51，6-166
DInfo 数据类型 DInfo data type 7-50
DXInfo 数据类型 DXInfo data type 7-50 至 7-51
DelMCEntries 过程 DelMCEntries procedure。参见 DeleteMCEntries 过程
DelMenuItem 过程 DelMenuItem procedure。参见 DeleteMenuItem 过程
DeleteMCEntries 过程 DeleteMCEntries procedure 3-147
DeleteMenu 过程 DeleteMenu procedure 3-109 至 3-110。参见 DeleteMenuItem 过程
DeleteMenuItem 过程 DeleteMenuItem procedure 3-127 至 3-128
DeskHook 全局变量 DeskHook global variable 4-75
DeskPattern 全局变量 DeskPattern global variable 4-113
DialogCopy 过程 DialogCopy procedure 6-133
DialogCut 过程 DialogCut procedure 6-132 至 6-133
DialogDelete 过程 DialogDelete procedure 6-134
DialogPaste 过程 DialogPaste procedure 6-133 至 6-134
DialogPeek 数据类型 DialogPeek data type 6-102
DialogPtr 数据类型 DialogPtr data type 6-102
DialogRecord 数据类型 DialogRecord data type 6-101 至 6-102
DialogSelect 函数 DialogSelect function 2-29，6-80，6-89 至 6-93，6-139 至 6-141
DisableItem 过程 3-58 至 3-59，3-131， DisableItem procedure 3-58 至 3-59，3-131 至 3-132
DispMCInfo 过程 DispMCInfo procedure。参见 DisposeMCInfo 过程
DisposDialog 过程 DisposDialog procedure。参见 DisposeDialog 过程
DisposeControl 过程 DisposeControl procedure 5-108
DisposeDialog 过程 DisposeDialog procedure 6-120
DisposeMCInfo 过程 DisposeMCInfo procedure 3-145
DisposeMenu 过程 DisposeMenu procedure 3-140
DisposeWindow 过程 DisposeWindow procedure 4-105 至 4-106
DlgCopy 过程 DlgCopy procedure。参见 DialogCopy 过程
DlgCut 过程 DlgCut procedure。参见 DialogCut 过程
DlgDelete 过程 DlgDelete procedure。参见 DialogDelete 过程
DlgFont 全局变量 DlgFont global variable 6-172
DlgPaste 过程 DlgPaste procedure。参见 DialogPaste 过程
DragControl 过程 DragControl procedure 5-99 至 5-100
DragGrayRgn 函数 DragGrayRgn function 4-96 至 4-98
DragHook 全局变量 DragHook global variable 4-98，4-100，4-102，4-104
DragWindow 过程 DragWindow procedure 4-44，4-94 至 4-95
DrawControls 过程 DrawControls procedure 5-87 至 5-88
DrawDialog 过程 DrawDialog procedure 6-142
DrawGrowIcon 过程 DrawGrowIcon procedure 4-12，4-39，4-86 至 4-87
DrawMenuBar 过程 DrawMenuBar procedure 3-51，3-58，3-113
DrawNew 过程 DrawNew procedure 4-117
Draw1Control 过程 Draw1Control procedure 5-88

E

EnableItem 过程 EnableItem procedure 3-58 至 3-59，3-131
EndUpdate 过程 EndUpdate procedure 2-47，

2-49，4-50，4-107
EraseRect 过程　EraseRect procedure 4-103
ErrorSound 过程　ErrorSound procedure 6-104
EvQEl 数据类型　EvQEl data type 2-83
EventAvail 函数　EventAvail function 2-21，2-88 至 2-89
EventRecord 数据类型　EventRecord data type 2-79 至 2-81

F

FInfo数据类型　FInfo data type 7-34，7-47至7-49
FSpGetFInfo 函数　FSpGetFInfo function 7-34，7-49
FSpSetFInfo 函数　FSpSetFInfo function 7-34，7-49
FXInfo 数据类型　FXInfo data type 7-49
FindControl 函数　FindControl function 5-33 至 5-35，5-44，5-53 至 5-54，5-58，5-89 至 5-90
FindDItem 函数　FindDItem function。参见 FindDialogItem函数
FindDialogItem 函数　FindDialogItem function 6-125
FindFolder 函数　FindFolder function 7-54至7-56
FindWindow 函数　FindWindow function 4-12，4-44，4-91 至 4-93，5-32 至 5-33
　和激活事件　and activate events 2-51
　和鼠标事件　and mouse events 2-33至2-37，4-42 至 4-44
Finder 标志　Finder flags 7-33 至 7-34，7-47 至 7-48
Finder 信息，目录文件中的——　Finder information，in the catalog file 7-32 至 7-34，7-46 至 7-51
Finder
　的应用程序界面　application interface to 7-3 至 7-78
　卷的数据库　database for a volume 7-45 至 7-46
　使用的数据结构　data structures used by 7-46 至 7-51
　定义　defined 7-3至7-5
　和进程管理器　and Process Manager 7-25
　和文件管理器的关系　relation至File Manager 7-3
　和资源管理器　and Resource Manager 7-3
　使用的资源　resources used by 7-6至7-32，7-56至7-70。参见应用程序未找到消息字符串资源；文件参考资源；图标列表资源；大8位彩色图标资源；大4位彩色图标资源；未找到应用程序名字字符串资源；署名资源；小8位彩色图标资源；小4位彩色图标资源；小图标列表资源 版本资源的例程　routines for 7-51至7-56的用户界面准则　user interface guidelines for 7-13，7-15
FlashMenuBar 过程　FlashMenuBar procedure 3-141 至 3-142
FlushEvents 过程　FlushEvents procedure 2-93至2-94
FrontWindow 函数　FrontWindow function 4-93 至 4-94

G

GetAlertStage 函数　GetAlertStage function 6-66，6-112
GetAlrtStage 函数　GetAlrtStage function。参见 GetAlertStage 函数
GetAuxCtl 函数　GetAuxCtl function。参见 GetAuxiliaryControlRecord 函数
GetAuxWin 函数　GetAuxWin function 4-115 至 4-116
GetAuxiliaryControlRecord 函数　GetAuxiliary-ControlRecord function 5-107
GetCRefCon 函数　GetCRefCon function。参见 GetControlReference 函数
GetCTitle 过程　GetCTitle procedure。参见 GetControlTitle 过程
GetCVariant 函数　GetCVariant function。参见 GetControlVariant 函数
GetCWMgrPort 过程　GetCWMgrPort procedure 4-113 至 4-114
GetCaretTime 函数　GetCaretTime function 2-113
GetControlAction 函数　GetControlAction function 5-106
GetControlMaximum 函数　GetControlMaximum function 5-41 至 5-42，5-61，5-104
GetControlMinimum 函数　GetControlMinimum function 5-103 至 5-104
GetControlReference 函数　GetControlReference function 5-105

GetControlTitle 过程 GetControlTitle procedure 5-104至5-105
GetControlValue 函数 GetControlValue function 5-38至5-39，5-41至5-42，5-44，5-53至5-55，5-61，5-103
GetControlVariant 函数 GetControlVariant function 5-106至5-107
GetCtlAction 函数 GetCtlAction function。参见 GetControlAction 函数
GetCtlMax 函数 GetCtlMax function。参见 GetControlMaximum 函数
GetCtlMin 函数 GetCtlMin function。参见 GetControlMinimum 函数
GetCtlValue 函数 GetCtlValue function。参见 GetControlValue 函数
GetDItem 过程 GetDItem procedure。参见 GetDialogItem 过程
GetDblTime 函数 GetDblTime function 2-113
GetDialogItem 过程 GetDialogItem procedure 6-50，6-121至6-122
GetDialogItemText 过程 GetDialogItemText procedure 6-50，6-130至6-131
GetEvQHdr 函数 GetEvQHdr function 2-100
GetGray 函数 GetGray function 3-91
GetGrayRgn 函数 GetGrayRgn function 4-113
GetIText 过程 GetIText procedure。参见 GetDialogItemText 过程
GetItem 过程 GetItem procedure。参见 GetMenuItemText 过程
GetItemCmd 过程 GetItemCmd procedure 3-138至3-139
GetItemIcon 过程 GetItemIcon procedure 3-137
GetItemMark 过程 GetItemMark procedure 3-135
GetItemStyle 过程 GetItemStyle procedure 3-133至3-134
GetKeys 过程 GetKeys procedure 2-110
GetMBarHeight 函数 GetMBarHeight function 3-113
GetMCEntry 函数 GetMCEntry function 3-145至3-146
GetMCInfo 函数 GetMCInfo function 3-52，3-143至3-144
GetMHandle 函数 GetMHandle function。参见 GetMenuHandle 函数
GetMenu 函数 GetMenu function 3-55，3-106至3-108
GetMenuBar 函数 GetMenuBar function 3-112
GetMenuHandle 函数 GetMenuHandle function 3-122
GetMenuItemText 过程 GetMenuItemText procedure 3-80，3-132
GetMouse 过程 GetMouse procedure 2-108
GetNewCWindow 函数 GetNewCWindow function 4-28，4-76至4-77
GetNewControl 函数 GetNewControl function 5-15至5-18，5-24至5-25，5-81至5-82
GetNewDialog 函数 GetNewDialog function 6-113至6-115
GetNewMBar 函数 GetNewMBar function 3-50至3-52，3-111至3-112
GetNewWindow 函数 GetNewWindow function 4-28，4-78至4-79
GetNextEvent 函数 GetNextEvent function 2-21，2-89至2-90
GetOSEvent 函数 GetOSEvent function 2-97至2-98
GetPortNameFromProcessSeriealNumber 函数 GetPortNameFromProcessSerialNumber function 2-107
GetProcessInformation 函数 GetProcessInformation function 2-74
GetProcessSerialNumberFromPortName 函数 GetProcessSerialNumberFromPortName function 2-73，2-82，2-106
GetSpecificHighLevelEvent 函数 GetSpecificHighLevelEvent function 2-71，2-77，2-92至2-93
GetWMgrPort 过程 GetWMgrPort procedure 4-114
GetWRefCon 函数 GetWRefCon function 4-111
GetWTitle 过程 GetWTitle procedure 4-86
GetWVariant 函数 GetWVariant function 4-24，4-112
GetWindowPic 函数 GetWindowPic function 4-110
GlobalToLocal 过程 GlobalToLocal procedure 5-34
GrayRgn 全局变量 GrayRgn global variable 4-16，4-95，4-113，4-116
GrowWindow 函数 GrowWindow function 4-57至4-59，4-99至4-100

H

HMGetHelpMenuHandle 函数 HMGetHelpMenuHandle function 3-67至3-68，3-81，3-123至3-124
HMSetMenuResID 过程 HMSetMenuResID procedure 6-71

HiWord 函数 HiWord function 4-58
HideControl 过程 HideControl procedure 5-39 至 5-40，5-65 至 5-67，5-97
HideDItem 过程 HideDItem procedure。参见 HideDialogItem 过程
HideDialogItem 过程 HideDialogItem procedure 6-123 至 6-124
HideWindow 过程 HideWindow procedure 4-61，4-89
HighLevelEventMsg 数据类型 HighLevelEventMsg data type 2-72，2-82
HiliteControl 过程 HiliteControl procedure 5-98 至 5-99，6-13 至 6-14，6-50 至 6-51，6-74，6-98
HiliteMenu 过程 HiliteMenu procedure 3-71，3-79，3-119 至 3-120
HiliteWindow过程 HiliteWindow procedure 4-90

I

IPCListPorts 函数 IPCListPorts function 2-74
InitDialogs 过程 InitDialogs procedure 6-103 至 6-104
InitMenu 过程 InitMenus procedure 3-103 至 3-104
InitProcMenu 过程 InitProcMenu procedure 3-104
InitWindows 过程 InitWindows procedure 4-75
InsMenuItem 过程 InsMenuItem procedure。参见 InsertMenuItem 过程
InsertMenu 过程 InsertMenu procedure 3-55，3-108 至 3-109
InsertMenuItem 过程 InsertMenuItem procedure 3-64 至 3-67，3-126 至 3-127
InsertResMenu 过程 InsertResMenu procedure 3-69，3-129 至 3-130
InvalMenuBar 过程 InvalMenuBar procedure 3-114
InvalRect 过程 InvalRect procedure 4-107 至 4-108
InvalRgn 过程 InvalRgn procedure 4-59，4-108
IsDialogEvent 函数 IsDialogEvent function 2-29，6-77，6-138 至 6-139
I型光标 I-beam cursor 2-63

K

KeyMap 数据类型 KeyMap data type 2-110
KeyTrans 函数 KeyTrans function。参见 KeyTranslate 函数
KeyTranslate 函数 KeyTranslate function 2-110 至 2-111，3-78
KillControl 过程 KillControls procedure 5-108 至 5-109

L

LoWord 函数 LoWord function 4-58
LocalAndRemoteHLEvents 标志 localAndRemoteHLEvents flag 2-119

M

MBarEnable 全局变量 MBarEnable global variable 3-121
MCEntry 数据类型 MCEntry data type 3-99
MCTableHandle 数据类型 MCTableHandle data type 3-99
MCTable 数据类型 MCTable data type 3-99
Macintosh 工具箱 Macintosh Toolbox
    提供的功能 features provided by 1-4
    介绍 introduction to 1-3 至 1-16
MenuChoice 函数 MenuChoice function 3-118 至 3-119
MenuHandle 数据类型 MenuHandle data type 3-95
MenuInfo 数据类型 MenuInfo data type 3-96 至 3-97
MenuKey 函数 3 MenuKey function 3-71，3-77 至 3-78，3-117 至 3-118
MenuSelect 函数 MenuSelect function 3-8，3-70，3-72，3-73，3-78 至 3-79，3-115 至 3-117
ModalDialog 过程 ModalDialog procedure 6-84，6-135 至 6-137
    和插入磁盘事件 and disk-inserted events 2-56
    和菜单 and menus 3-86
MoveControl 过程 MoveControl procedure 5-67 至 5-70，5-97 至 5-98
MoveWindow 过程 MoveWindow procedure 4-95 至 4-96

MultiFinder 2-7
MyAction 过程 MyAction procedure 5-116至5-117
MyAlertSound 过程 MyAlertSound procedure 6-144至6-145
MyControl 函数 MyControl function 5-110至5-115
MyEventFilter 函数 MyEventFilter function 6-145至6-147
MyIndicatorAction 过程 MyIndicatorAction procedure 5-117
MyItem 过程 MyItem procedure 6-143至6-144

N

NewCDialog 函数 NewCDialog function。参见 NewColorDialog 函数
NewCWindow 函数 NewCWindow function 4-79至4-82
NewColorDialog 函数 NewColorDialog function 6-115至6-118
NewControl 函数5 NewControl function 5-82至5-85
NewDialog 函数 NewDialog function 6-118至6-119
NewMenu 函数 NewMenu function 3-105至3-106
NewWindow 函数 NewWindow function 4-82至4-85
NoteAlert 函数 NoteAlert function 6-110
NoteAlert 函数创建"注意"示警框 creating with the NoteAlert function 6-110

O

OSEventAvail 函数 OSEventAvail function 2-98至2-99
OldContent 全局变量 OldContent global variable 4-117
OldStructure 全局变量 OldStructure global variable 4-117
OpenDeskAcc 函数 OpenDeskAcc function 3-21，3-80至3-81
Option键 Option key 2-20

P

PBGetCatInfo 函数 PBGetCatInfo function 7-34，7-49
PBMountVol 函数 PBMountVol function 2-55
PPCBrowser 函数 PPCBrowser function
  填写一个目标标识符 记录 filling in a target ID record 2-77
  查找一个指定的端口 finding a specific port 2-75
  投寄一个高级事件 posting a high-level event 2-74至2-75
PPC（程序间通信）工具箱 PPC (Program-to-Program Communications) Toolbox 1-16，2-7
PPostEvent 函数 PPostEvent function 2-103至2-104
PaintBehind 过程 PaintBehind procedure 4-118至4-119
PaintOne 过程 PaintOne procedure 4-118
PaintWhite 全局变量 PaintWhite global variable 4-118，4-119
ParamText 过程 ParamText procedure 6-47，6-129至6-130
PinRect 函数 PinRect function 4-99
PopUpMenuSelect 函数 PopUpMenuSelect function 3-57，3-120至3-121
PostEvent 函数 PostEvent function 2-104至2-105
PostHighLevelEvent 函数 PostHighLevelEvent function 2-73至2-77，2-101至2-103

Q

QHdr 数据类型 QHdr data type 2-83
QuickDraw QuickDraw 1-14。参见 彩色 QuickDraw
QuickDraw 的坐标系 coordinate systems in 4-17至4-18

R

RGBColor 数据类型 RGBColor data type 5-79
ResetAlertStage过程 ResetAlertStage procedure 6-112
ResetAlrtStage 过程 ResetAlrtStage procedure。参见 ResetAlertStage 过程。
ResolveAliasFile函数 ResolveAliasFile function 7-52至7-53
ResumeProc 全局变量 ResumeProc global variable 6-172

S

ScrollRect 过程 ScrollRect procedure 5-46至5-48，5-56至5-57
SelIText 过程 SelIText procedure。参见SelectDialogItemText过程
SelectDialogItemText过程 SelectDialogItemText procedure 6-131至6-132
SelectWindow 过程 SelectWindow procedure 2-51，4-87至4-88
SendBehind 过程 SendBehind procedure 4-91
SetCRcon 过程 SetCRefCon procedure。参见SetControlReference 过程
SetCTitle 过程 SetCTitle procedure。参见SetControlTitle过程
SetControlAction 过程 SetControlAction procedure 5-102
SetControlColor过程 SetControlColor procedure 5-101
SetControlMaximum 过程 SetControlMaximum procedure 5-41至5-43，5-70，5-95至5-96
SetControlMinimum 过程 SetControlMinimum procedure 5-95
SetControlReference过程 SetControlReference procedure 5-105至5-106
SetControlTitle过程 SetControlTitle procedure 5-96
SetControlValue 过程 SetControlValue procedure 5-38至5-39，5-42至5-43，5-44，5-61，5-94至5-95
SetCtlAction过程 SetCtlAction procedure。参见SetControlAction 过程
SetCtlColor 过程 SetCtlColor procedure。参见SetControlColor过程
SetCtlMax 过程 SetCtlMax procedure。参见SetControlMaximum 过程
SetCtlMin 过程 SetCtlMin procedure。参见SetControlMinimum 过程
SetDAFont 过程 SetDAFont procedure。参见SetDialogFont 过程
SetDItem 过程 SetDItem procedure。参见SetDialogItem 过程
SetDeskCPat 过程 SetDeskCPat procedure 4-112至4-113
SetDialogFont 过程 SetDialogFont procedure 6-105
SetDialogItem 过程 SetDialogItem procedure 6-122至6-123
SetDialogItemText 过程 SetDialogItemText procedure 6-131
SetEventMask 过程 SetEventMask procedure 2-99至2-100
SetIText 过程 SetIText procedure。参见SetDialogItemText 过程
SetItem 过程 SetItem procedure。参见SetMenuItemText 过程
SetItemCmd 过程 SetItemCmd procedure 3-139至3-140
SetItemIcon 过程 SetItemIcon procedure 3-62至3-64，3-137至3-138
SetItemMark 过程 SetItemMark procedure 3-61至3-62，3-135至3-136
SetItemStyle 过程 SetItemStyle procedure 3-60，3-134
SetMCEntries 过程 SetMCEntries procedure 3-146至3-147
SetMCInfo 过程 SetMCInfo procedure 3-52，3-144
SetMenuBar 过程 SetMenuBar procedure 3-50至3-51，3-112至3-113
SetMenuFlash 过程 SetMenuFlash procedure 3-142
SetMenuItemText 过程 SetMenuItemText procedure 3-59至3-60，3-133
SetOrigin 过程 SetOrigin procedure 5-44至5-45，5-63至5-64
SetPort 过程 SetPort procedure 6-62
SetWRefCon 过程 SetWRefCon procedure 4-111
SetWTitle 过程 SetWTitle procedure 4-85
SetWinColor 过程 SetWinColor procedure 4-114至4-115
SetWindowPic 过程 SetWindowPic procedure 4-110
Shift 键 Shift key 2-20
ShortenDITL 过程 ShortenDITL procedure 6-127至6-128
ShowControl 过程 ShowControl procedure 5-39至5-40，5-66至5-67，5-86
ShowDItem 过程 ShowDItem procedure。参见ShowDialogItem 过程
ShowDialogItem 过程 ShowDialogItem procedure 6-124
ShowHide 过程 ShowHide procedure 4-89至4-90
ShowWindow 过程 ShowWindow procedure 4-88，6-61
SizeControl 过程 SizeControl procedure 5-67至5-70，5-98
SizeWindow 过程 SizeWindow procedure 4-60，4-101
StillDown 函数 StillDown function 2-109
StopAlert 函数 StopAlert function 6-109

Style 数据类型 Style data type 3-134
StyleItem 数据类型 StyleItem data type 3-134
SystemClick 过程 SystemClick procedure 2-87, 2-94 至 2-95, 4-43, 4-44
SystemEdit 函数 SystemEdit function 3-122
SystemEvent 函数 SystemEvent function 2-96
SystemMenu 过程 SystemMenu procedure 3-121
SystemTask 过程 SystemTask procedure 2-25, 2-87, 2-95 至 2-96

T

TEPinScroll 过程 TEPinScroll procedure 5-55 至 5-56
TEUpdate 过程 TEUpdate procedure 5-64 至 5-65
TargetID 数据类型 TargetID data type 2-72, 2-81
TestControl 函数 TestControl function 5-93
TextEdit
  和对话管理器 and Dialog Manager 6-79 至 6-80
  处理文字 handling text 1-14 至 1-15
  和鼠标事件 and mouse events 2-37
TickCount 函数 TickCount function 2-112
Ticks 全局变量 Ticks global variable 2-112
TopMenuItem 全局变量 TopMenuItem global variable 3-151
TrackBox 函数 TrackBox function 4-44, 4-101 至 4-102
TrackControl 函数 TrackControl function 5-35 至 5-37, 5-44, 5-53 至 5-55, 5-58, 5-90 至 5-92, 6-78
  和弹出式菜单 and pop-up menus 3-56
TrackGoAway 函数 TrackGoAway function 4-44, 4-103 至 4-104
TrueType 字形 TrueType font 。参见字形

U

UpdateControl 过程 UpdateControls procedure 5-86 至 5-87
UpdateDialog 过程 UpdateDialog procedure 6-142 至 6-143
UpdtControl 过程 UpdtControl procedure。参见 UpdateControl 过程
UpdtDialog 过程 UpdtDialog procedure。参见 UpdateDialog 过程

V

ValidRect 过程 ValidRect procedure 4-58, 4-108 至 4-109
ValidRgn 过程 ValidRgn procedure 4-109

W

WStateData 数据类型 WStateData data type 4-33, 4-54, 4-70 至 4-71
WaitMouseUp 函数 WaitMouseUp function 2-109
WaitNextEvent 函数 WaitNextEvent function 2-85 至 2-88
  引入 introduced 2-22 至 2-24
  和多任务 and multitasking 2-60
WinCTab 数据类型 WinCTab data type 4-71 至 4-72
WindowList 全局变量 WindowList global variable 4-74
WindowPeek 数据类型 WindowPeek data type 4-20, 4-65
WindowPtr 数据类型 WindowPtr data type 4-19, 4-65
WindowRecord 数据类型 WindowRecord data type 4-19, 4-65, 4-69

Z

ZoomWindow 过程 ZoomWindow procedure 4-54, 4-57, 4-102 至 4-103

a

acceptAppDieEvents 标志 acceptAppDiedEvents flag 2-118
acceptSuspendResumeEvents 标志 acceptSuspendResumeEvents flag 2-117
altDBoxProc 窗口类型 altDBoxProc window type 4-9

c

canBackground 标志 canBackground flag2-117
color 标志 color flag 7-48

d

dBoxProc 窗口类型 dBoxProc window type 2-16，4-9，6-25
disabled 常数 disabled constant 3-8
documentProc 窗口类型 documentProc window type 4-10
doesActivateOnFGSwitch 标志 doesActivate-OnFGSwitch flag 2-117
drop文件夹及其替身类型 drop folder, alias type for 7-40

e

enabled 常数 enabled constant 3-8

g

getFrontClicks 标志 getFrontClicks flag 2-118

h

hasBeenInited 标志 hasBeenInited flag 7-48
hasBundle 标志 7-17至7-18， hasBundle flag 7-34
hasCustomIcon 标志 hasCustomIcon flag 7-17至7-18，7-34
hasNoINITS 标志 hasNoINITS flag 7-48

i

is32BitCompatible 标志 is32BitCompatible flag 2-118
isAlias 标志 isAlias flag 7-40
isHighLevelEventAware 标志 isHighLevelEv-entAware flag 2-119
isInvisible 标志 isInvisible flag 7-34
isOnDesk 标志 isOnDesk flag 7-48
isShared 标志 isShared flag 7-34，7-48
isStationery 标志 isStationery flag 7-34
isStationeryAware 标志 isStationeryAware flag 2-119

m

movableDBoxProc 窗口类型 movableDBoxProc window type 3-85，4-9，6-25
nameLocked 标志 nameLocked flag 7-18，7-34

n

noGrowDocProc 窗口类型 noGrowDocProc window type 4-10，6-25
onlyBackground 标志 onlyBackground flag 2-117

p

pCDeskPat 参数—RAM 位标志 pCDeskPat paramet-er-RAM bit flag 4-113
plainDBoxProc 窗口类型 plainDBoxProc window type 4-9
popupPrivateData 数据类型 popupPrivateData data type 5-77

r

rDocProc 窗口类型 rDocProc window type 4-10
requiresSwitchLaunch 标志 requiresSwitchLaunch flag 7-48

u

useTextEditService 标志 useTextEditServices flag 2-119

z

zoomDocProc 窗口类型 zoomDocProc window type 4-8
zoomNoGrow 窗口类型 zoomNoGrow window type 4-10

符号

‘ **** ’ 文件类型 ’****’ file type 7-20
‘****’ 文件类型 ’****’ file type 7-20
‘????’ 文件类型 ’????’ file type 7-20

A

’ALRT’ 资源类型 ’ALRT’ resource type。参见警告

资源
'APPL' 文件类型 'APPL' file type 7-10

B

'BNDL' 资源类型 'BNDL' resource type 7-20 至 7-25，7-65 至 7-68

C

'CDEF' 资源类型 'CDEF' resource type 5-14，5-109 至 5-115
'CNTL' 资源类型 'CNTL' resource type
    的实例 example of 5-18，5-21，5-23，5-26
    的Rez 输入格式 Rez input format for 5-18至 5-24，5-25 至 5-28
    的Rez 输出格式 Rez output format for 5-118 至 5-121

D

'DITL' 资源类型 'DITL' resource type。参见项目列表资源
'DLOG' 资源类型 'DLOG' resource type。参见对话资源
'DRVR' 资源类型 'DRVR' resource type 3-20，3-68，3-129，3-130

F

'FKEY' 资源类型 'FKEY' resource type 2-39，2-86，3-118
'FOND' 资源类型 'FOND' resource type 3-69
'FONT' 资源类型 'FONT' resource type 3-69
'FREF' 资源类型 'FREF' resource type 7-18 至 7-25，7-64 至 7-65

I

'ICN#' 资源类型 'ICN#' resource type
    创建 creating 7-11至7-15
    的实例 example of 7-14 至 7-15
    的Rez 输出格式 Rez output format for 7-57至 7-58
    在束资源中指定 specifying in a bundle resource 7-20
'ICON' 资源类型 'ICON' resource type 7-63
    指定菜单项目的 specifying a menu item's 3-46，3-62，3-154
'INIT' 文件类型 'INIT' file type 7-44

K

'KCHR' 资源类型 'KCHR' resource type 2-39 至 2-42，3-17
'KMAP' 资源类型 'KMAP' resource type 2-21，2-39 至 2-42

M

'MBAR' 资源类型 'MBAR' resource type 3-50，3-155
'MBDF' 资源类型 'MBDF' resource type 3-9，3-87，3-104
'MDEF' 资源类型 'MDEF' resource type 3-87，3-157
'MENU' 资源类型 'MENU' resource type 3-42 至 3-48，3-152 至 3-154

S

'SICN' 资源类型，指定菜单项目的 'SICN' 资源类型 'SICN' resource type，specifying a menu item's 3-62，3-154
'SIZE' 资源类型 'SIZE'resource type 2-115 至 2-119，7-35
    创建 creating 2-30至2-32
    定义 defined 2-30 至 2-32，2-115 至 2-119
    标志，定义的…标志 flags，defined 2-116 至 2-119
    和空事件 and null events 2-9，2-57
    的示例Rez 输入格式 sample Rez input 2-31
    计划…的可选标志 scheduling option flags 2-16
    设定高级事件的…标志 setting flags for high-level events 2-68
    设定…的标志 setting flags of 2-9，2-11，2-16，2-51

'STR' 资源类型 'STR' resource type 3-60, 7-27 至 7-30
'STR#' 资源类型 'STR#' resource type 3-60, 3-65 至 3-67

T

'TEXT' 文件类型 'TEXT' file type 7-10

W

'WDEF' 资源类型 'WDEF' resource type 4-22, 4-120, 4-127
'WIND' 资源类型 'WIND' resource type 4-22, 4-25 至 4-27, 4-124 至 4-127

a

'actb' 资源类型 'actb' resource type 6-157 至 6-158
'alis' 资源类型 'alis' resource type 7-40
'amnu' 文件夹类型 'amnu' folder type 7-54
'appe' 文件类型 'appe' file type 7-44

c

'cctb' 资源类型 'cctb' resource type 5-121 至 5-123
'cicn' 资源类型 'cicn' resource type 7-64
    指定菜单项目的 specifying a menu item's 3-62, 3-154
'ctrl' 文件夹类型 'ctrl' folder type 7-54

d

'dctb' 资源类型 'dctb' resource type 6-156 至 6-157
'desk' 文件夹类型 'desk' folder type 7-54
'disk' 文件类型 'disk' file type 7-20
'drop' 替身类型 'drop' alias type 7-40

e

'empt' 文件夹类型 'empt' folder type 7-54

f

'faam' 替身类型 'faam' alias type 7-40
'fact' 替身类型 'fact' alias type 7-40
'fadr' 替身类型 'fadr' alias type 7-40
'faet' 替身类型 'faet' alias type 7-40
'faex' 替身类型 'faex' alias type 7-40
'famn' 替身类型 'famn' alias type 7-40
'fapf' 替身类型 'fapf' alias type 7-40
'fapn' 替身类型 'fapn' alias type 7-40
'fash' 替身类型 'fash' alias type 7-40
'fast' 替身类型 'fast' alias type 7-40
'fasy' 替身类型 'fasy' alias type 7-40
'fdrp' 替身类型 'fdrp' alias type 7-40
'ffil' 文件类型 'ffil' file type 7-37
'fld#' 替身类型 'fld#' resource type 7-55
'flpy' 替身类型 'flpy' alias type 7-40
'fold' 文件类型 'fold' file type 7-20
'font' 文件夹类型 'font' folder type 7-54

h

'hdsk' 替身类型 'hdsk' alias type 7-40
'hmmu' 资源类型 'hmmu' resource type 3-44, 3-68

i

'icl4' 资源类型 'icl4' resource type
    创建 creating 7-11, 7-24
    的Rez 输出格式 Rez output format for 7-59 至 7-60
'icl8' 资源类型 'icl8' resource type
    创建 creating 7-11, 7-24
    的Rez 输出格式 Rez output format for 7-61 至 7-62
'ics#' 资源类型 'ics#' resource type
    创建 creating 7-11, 7-15, 7-24
    的Rez 输出格式 Rez output format for 7-58 至 7-59
'ics4' 资源类型 'ics4' resource type

创建 creating 7-11，7-24
的Rez 输出格式 Rez output format for 7-60至7-61
'ics8' 资源类型 'ics8' resource type
创建 creating 7-11，7-24
的Rez 输出格式 Rez output format for 7-62至7-63
'ictb' 资源类型 'ictb' resource type 6-158 至 6-164
'ifil' 文件类型 'ifil' file type 7-37
'itlc' 资源类型 'itlc' resource type 3-32，3-33
'itlk' 资源类型 'itlk' resource type 2-41

k

'kfil' 文件类型 'kfil' file type 7-37

m

'macs' 文件夹类型 'macs' folder type 7-54
'mctb' 资源类型 'mctb' resource type 3-52，3-99，3-143，3-155 至 3-157

p

'pref' 文件夹类型 'pref' folder type 7-54
'prnt' 文件夹类型 'prnt' folder type 7-54
'scri' 文件类型 'scri' file type 7-44

s

'sfil' 文件类型 'sfil' file type 7-37
'srvr' 替身类型 'srvr' alias type 7-40
'strt' 文件夹类型 'strt' folder type 7-54

t

'temp' 文件夹类型 'temp' folder type 7-54
'trsh' 替身类型 'trsh' alias type 7-40
'trsh' 文件夹类型 'trsh' folder type 7-54

v

'vers' 资源类型 'vers'resource type 7-31 至 7-32，7-69 至 7-70

w

'wctb' 资源类型 'wctb' resource type 4-71，4-127 至 4-129

A

安装的文件夹及其替身类型 mounted folder, alias type for 7-40
安装和删除 installing and removing 7-41
按钮 buttons
活动按钮 active 5-11至5-13
控制部件定义标识符 control definition ID for 5-14
创建，在窗口中 creating, in windows 5-17 至 5-20
默认 default 6-10至6-11，6-22，6-30至6-31，6-56至6-61
定义的 defined 5-5
中的事件 events in 5-31至5-37，6-78
不活动 inactive 5-11至5-13
的部位代码 part code for 5-32
在示警框或对话框中说明 specifying in alert or dialog boxes 6-29至6-30，6-153
的标题 titles for 6-37至6-39

B

版本请求中的区域代码 region codes, in version resources 7-31
版本资源 version resources 7-31至7-32，7-69至7-70
帮助菜单 Help menu 3-29至3-31
从示警框和对话框存取 accessing from alert and dialog boxes 6-68至6-69
向…添加项目 adding items to 3-67至3-68
处理 handling 3-81
的“隐藏旁述”命令 Hide Balloons command 6-68
的“显示旁述”命令 Show Balloons command 6-68
帮助项目，在示警框和对话框中指定的——

help items，specifying in alert or dialog boxes 6-29至6-30，6-155至6-156
编辑菜单　Edit menu 3-24至3-26
从示警框和对话框中存取　accessing from alert and dialog boxes 6-68至6-73
调整　adjusting 3-75至3-77
的“不能取消”命令　Can't Undo command 3-25
的“清除”命令　Clear command 3-25，6-69
的“拷贝”命令　Copy command 3-25，6-69
的“创建版本”命令　Create Publisher command 3-25
的“剪切”命令　Cut command 3-25，6-69
的“隐藏夹纸板”命令　Hide Clipboard command 3-25
的“贴”命令　Paste command 3-25，6-69
的“版本选项”命令　Publisher Options command 3-25
的“全选”命令　Select All command 3-25
的“显示夹纸板”命令　Show Clipboard command 3-25
的“订户选项”命令　Subscriber Options command 3-25
的 SubscribeTO 命令　Subscribe至 command 3-25
的“取消”命令　Undo command 3-25，6-69
变量代码　variation codes
控制部件的…，定义的控制部件的… for controls，defined 5-14
确定…，确定控制部件的… determining，for controls 5-106至5-107
窗口的…　for windows 4-11，4-112，4-120
标签菜单命令　Label menu commands 7-16
标题　titles
按钮、复选框和单选按钮的标题　of buttons，checkboxes，and radio buttons 6-37至6-40，6-153
改变标题，改变控制部件的标题 changing，for controls 5-96
确定标题，确定控制部件的标题 determining，for controls 5-104至5-105
无模式对话框和可移动模式对话框的标题　for modeless and movable modal dialog boxes 6-25
指定控制部件的标题　specifying for controls 5-18，5-19至5-20，5-25，5-83，5-85，5-120，5-121
标题栏　title bar 4-5
标准控制定义函数　standard control definition function 5-14
标准文件对话框，标准文件对话框和插入磁盘事件 standard file dialog boxes，and disk-inserted events 2-55
标准文件软件包，标准文件软件包和插入磁盘事件 Standard File Package，and disk-inserted events 2-55至2-56
拨盘控制部件　dial controls 5-11
“不能恢复”命令（编辑菜单）　Can't Undo command (Edit menu)　3-24，3-25
部位代码　part codes 5-31至5-32

C

彩色　Color QuickDraw
检查　checking for
创建窗口时　when creating a window 4-77
缩放窗口时　when zooming windows 4-54
和彩色窗口记录　and color window records 4-65
中的坐标系统　coordinate systems in 4-17至4-18
创建彩色窗口　creating color windows 4-20至4-21
和窗口管理器端口　and the Window Manager port 4-74
彩色图标　color icons 7-11，7-13至7-16
彩色图标资源　color icon resources 3-46，3-62，7-64
彩色图形端口　color graphics ports 4-20
菜单　menus。参见苹果菜单；应用程序菜单；编辑菜单；文件菜单；字体菜单；帮助菜单；键盘菜单；大小菜单；样式菜单
从示警框和对话框存取菜单　accessing from alert and dialog boxes 3-84至3-86，6-68至6-73
向菜单添加项目　adding items to 3-64至

3-70
调整菜单中的项目　adjusting items in 3-73至3-77
苹果菜单　Apple。参见苹果菜单
应用程序　Application。参见应用程序菜单
彩色菜单　color 3-11
创建菜单　creating 3-42至3-57，3-105至3-108
创建嵌套式菜单　hierarchical 3-53至3-56
创建弹出式菜单　pop-up 3-56至3-57，5-25至5-28，6-42至6-44
下拉式菜单　pull-down 3-42至3-56
停用示警框和对话框中的菜单　disabling for alert and dialog boxes 6-68至6-73
处理菜单　disposing of 3-112，3-140
编辑菜单　Edit。参见编辑菜单
文件菜单　File。参见文件菜单
字体菜单　Font。参见字形菜单
把菜单中的项目分组　grouping items in 3-14至3-16
帮助菜单　Help。参见帮助菜单
菜单的旁述　help balloons for 3-31
嵌套菜单　hierarchical 3-6，3-38，3-53至3-56
菜单中的项目　items in。参见菜单项目
键盘菜单　Keyboard。参见键盘菜单
菜单的等效组合键　keyboard equivalents for 3-16至3-19，3-154
菜单本地化　localizing 3-10
菜单　menu ID 3-152
弹出式菜单　pop-up。参见弹出式菜单
下拉式菜单　pull-down 3-6，3-42
菜单的资源标识符　resource ID 3-44，3-153
大小菜单　Size。参见大小菜单
式样菜单　Style。参见样式菜单
子菜单　submenus 3-6，3-53至3-56
系统处理的菜单　system-handled 3-19
菜单标题　titles of 3-7，3-10
菜单的类型　types of 3-6
菜单标识符　menu ID 3-40，3-152
菜单定义过程　menu definition procedure 3-148至3-151
和 A5 寄存器　and the A5 register 3-150
定义　defined 3-9
绘制彩色菜单的过程　drawing color menus 3-91
在菜单中处理滚动的过程　handling scrolling in menus 3-91，3-93
标准　standard 3-148
编写　writing 3-87至3-95
菜单管理器　Menu Manager 3-5至3-167
应用程序定义的…例程　applicationdefined routines for 3-148至3-151
中的数据类型　data structures in 3-95至3-102
保留的数据类型　data structures maintained by 3-40至3-41
和对话管理器　and Dialog Manager 6-68至6-73
初始化　initializing 3-103至3-104
的本地化准则　localization guidelines for 3-10，3-16至3-18，3-43
的例程名称，的例程名称的映射　routine names，mapping of 3-102
中的例程　routines in 3-102至3-147
测试…的可用性　testing for availability 3-42
的用户界面准则　user interface guidelines for 3-5至3-38
菜单记录　menu records 3-40，3-95至3-97
菜单句柄　menu handle 3-95，3-122
菜单列表　menu lists 3-40，3-97至3-98
菜单列表中的标记字符　marking character in a menu item 3-12
菜单命令　menu commands。参见菜单项目
“关于”(苹果菜单)　About (Apple menu) 3-21
“不能取消”(编辑菜单)　Can't Undo (Edit menu) 3-25
在Finder中改变色彩的　for changing colors in the Finder 7-16
“清除”(编辑菜单)　Clear (Edit menu) 3-25，6-68至6-69
“关闭”(文件菜单)　Close (File menu) 3-23
“拷贝”(编辑菜单)　Copy (Edit menu) 3-25，6-68至6-69
“创建版本”(编辑菜单)　Create Publisher (Edit menu) 3-25
“剪”(编辑菜单)　Cut (Edit menu) 3-25，6-68至6-69
“旁述”(帮助菜单)　Hide Balloons (Help menu) 6-68
“隐藏夹纸板”(编辑菜单)　Hide Clipboard (Edit menu) 3-25
“隐藏其它”(应用程序菜单)　Hide Others (Application menu) 3-33
的等效组合键　keyboard equivalents for 3-16至3-19，3-154

"新文稿"(文件菜单) New (File menu) 3-23
"无"(彩色菜单) None (Color menu) 7-16
"打开"(文件菜单) Open (File menu) 3-23
"其它"(大小菜单) Other (Size menu) 3-28
"页面设置"(文件菜单) Page Setup (File menu) 3-23
"贴"(编辑菜单) Paste (Edit menu) 3-25, 6-68至6-69, 7-17
"打印"(文件菜单) Print (File menu) 3-23
"版本选项"(编辑菜单) Publisher Options (Edit menu) 3-25
"退出"(文件菜单) Quit (File menu) 3-23
"存储"(文件菜单) Save (File menu) 3-23
"存储为"(文件菜单) Save As (File menu) 3-23
"全选"(编辑菜单) Select All (Edit menu) 3-25
"显示旁述"(帮助菜单) Show Balloons (Help menu) 3-30, 6-68
"显示剪贴板" Show Clipboard (Edit menu) 3-25
"引用选项"(编辑菜单) Subscriber Options (Edit menu) 3-25
"引用"(编辑菜单) Subscribe To (Edit menu) 3-25
"取消"(编辑菜单) Undo (Edit menu) 3-25, 6-68至6-69
菜单条 menu bars
从示警框和对话框存取 accessing from alert and dialog boxes 6-68至6-73
创建 creating 3-49至3-50
定义 defined 3-6, 3-10
处理…中的鼠标按下事件 handling mouse down events in 3-72
安装 installing 3-41
菜单条定义函数 menu bar definition function 3-9, 3-87, 3-104
菜单项目 menu items 3-12至3-16
累积组中的 in an accumulating group 3-15
添加至苹果菜单的 adding to Apple menu 3-68至3-69
添加至帮助菜单的 adding to Help menu 3-67至3-68
添加至菜单的 adding to menus 3-64至3-70
的外观 appearance of 3-12至3-14
改变 changing
的字形 font style of 3-60
的图标 icon of 3-62至3-64
的标记 mark of 3-61至3-62
的语系代码 script code of 3-62
的文字 text of 3-59至3-60
的特征 characteristics of 3-12至3-14
停用 disabling 3-8, 3-58至3-59, 3-74至3-77
的分隔线 divider 3-12
启用 enabling 3-8, 3-58至3-59, 3-74至3-77
的字形 font style of 3-47, 3-154
分组 grouping 3-14至3-16
处理用户选定的 handling user selection of 3-70至3-78
的图标 icon of 3-154
的编号 item numbers 3-40
的等效组合键 keyboard equivalents for 3-16至3-19, 3-154
标记…的特征 marking character of 3-154
中的元字符 metacharacters in 3-65
互斥组中的 in a mutually exclusive group 3-14
为…提供旁述 providing help balloons for 3-7
的语系代码 script code of 3-47, 3-62, 3-154
的语系 script of 3-12
指定…的特征 specifying the characteristics of 3-45至3-47
的子菜单 submenu for 3-47, 3-154
在旁述和用户文稿中使用的…术语 terms to use in help balloons and user documentation 3-8
的文字 text of 3-12至3-13
菜单项目文字的语系代码 script code, for menu item text 3-47, 3-62
菜单颜色信息表 menu color information table 3-52, 3-98至3-102
的菜单条入口 menu bar entry 3-100
的菜单项目入口 menu item entry 3-101
的菜单标题入口 menu title entry 3-101
资源 resources 3-155至3-157
菜单资源 menu resource 3-151至3-154

参考值　reference values

改变…，改变控制部件的…　changing，for controls 5-105至5-106

确定…，确定控制部件的…　determining，for controls 5-105

弹出式菜单的…　for pop-up menus 5-28

指定…，指定控制部件的…　specifying，for controls 5-18，5-25，5-84，5-120

操作系统事件　operating-system events

操作系统事件队列　Operating System event queue

操作系统事件管理器　Operating System Event Manager。参见事件管理器

插入符，闪烁间隔 3-24，3-25　caret，time between blinks of 2-113

查找和系统有关的目录　directories，finding system-related 7-53至7-56

程序间通信（PPC）工具箱　Program-to-Program Communications (PPC) Toolbox 1-16，2-7

出版数据　editions

为…创建图标　creating icons for 7-16

定义　defined 7-4

出版数据管理器　Edition Manager 2-14

“出版选项”命令（编辑菜单）　Publisher Options command (Edit menu) 3-25

处理操作系统事件　handling 2-58至2-67

处理空事件　handling 2-57

处理文字　handling 1-14至1-15

窗口　windows 4-3至4-129

激活窗口　activating 4-50至4-53

现用窗口　active 4-6

示警框和对话框窗口　alert boxes and dialog boxes

停用示警框和对话框后面的窗口　deactivating behind 6-64至6-66

作为窗口 类型的示警框和对话框　as types of 6-15至6-17

关闭窗口　closing 4-60至4-62，4-103至4-106

窗口 的颜色　color in 4-20至4-21，4-71至4-74

窗口 内容区域　content region 4-12

窗口 中的控制部件　controls in 4-14至4-15，5-82

创建窗口　creating 4-25至4-30，4-75至4-85

解除分配窗口　deallocating 4-61至4-62，4-104至4-106

定义窗口　defined 4-4

显示窗口　displaying 4-86至4-91

绘制窗口 的内容区域　drawing content region 4-39至4-40

窗口 中的事件　events in 4-21，4-22，4-41至4-53

窗口 的增长图象　grow image 4-57，4-87

隐藏窗口　hiding 4-62至4-64，4-89

非当前活动窗口　inactive 4-6至4-7

保留窗口 的更新区域　maintaining update region of 4-106至4-109

管理多个窗口　managing multiple 4-15至4-16，4-23至4-25

操纵窗口　manipulating

特征　characteristics 4-109至4-112

颜色　color 4-114至4-116

在桌面上操纵窗口　on the desktop 4-112至4-114

窗口 中的按下鼠标事件　mouse-down events in 4-42至4-45

移动窗口　moving 4-53，4-94至4-99

命名窗口　naming 4-85至4-86

用窗口管理器绘制窗口部分　parts drawn by the Window Manager 4-12至4-13

窗口定位　positioning 4-30至4-38

窗口中的区域　regions in 4-12至4-13

重新调整窗口 的大小　resizing 4-57至4-59

响应窗口 中的激活事件　responding to activate events in 2-50至2-55，4-50至4-53

从窗口中检索信息　retrieving information 4-91至4-94

滚动窗口　scrolling 5-9至5-10，5-43至5-70

显示窗口　showing 4-62至4-64，4-88

调整窗口 的大小　sizing 4-99至4-101

窗口 的标准状态　standard state 4-33，4-53，4-70

窗口 的结构区域　structure region 4-12

窗口 的类型　types of 4-8至4-10

更新窗口　updating 2-47至2-50，4-40至4-41，4-48至4-50，5-49至5-52，5-62至5-65

用户状态　user state 4-33，4-53，4-70

操纵窗口 的惯例　window-manipulation conventions 4-21

窗口 资源，定义窗口 资源　window resources，defining 4-25至4-27

缩放窗口　zooming 4-53至4-56，4-101至4-103

窗口的标准状态　standard state of a window 4-33，4-53，4-70
窗口的用户状态　user state of a window 4-33，4-53，4-70
窗口定义标识符　window definition IDs
创建窗口的…，在创建窗口中使用的… creating windows，used in 4-80，4-83至4-84
窗口定义函数　window definition functions
定义　defined 4-10
编写　writing 4-120至4-124
窗口定义函数的窗口定义标识符　and window definition functions 4-120
窗口管理器　Window Manager 4-3至4-129
应用程序定义的…例程　application-defined routine for 4-120至4-124
和控制管理器　and Control Manager 4-14至4-15，5-3
中的数据结构　data structures in 4-65至4-74
和对话管理器　and Dialog Manager 4-13至4-14，6-15至6-16
生成的事件　events generated by 2-9
全局变量　global variables 4-75
端口　port 4-74
中的资源　resources in 4-124至4-129
的例程　routines in 4-74至4-119
初始化…的例程　initializing4-74至4-75
中的低级例程　low-level routines 4-116至4-119
窗口管理器端口的——　of Window Manager port 4-75
窗口记录　window records 4-19至4-20，4-65至4-69
窗口框架　window frames 4-6，4-12至4-13，4-121至4-122
窗口类型　window types 4-8至4-11
窗口列表　window list 4-15，4-74
窗口起始位置　window origin
改变　changing 5-44至5-45，5-48至5-52
定义的　defined 5-45
窗口区域　window regions 4-12至4-13
窗口位置常数　window positioning constants 4-32
窗口颜色表　window color table 4-21，4-71至4-73
窗口中的图象增长　grow images，of windows 4-57，4-87
窗口状态数据记录　window state data record 4-33，4-54，4-70至4-71
窗口资源　window resources 4-22，4-25至4-27，4-124至4-127
窗口资源的窗口定义标识符　in window resources 4-26，4-125
“创建版本”命令（编辑菜单）　Create Publisher command　(Edit menu)　3-25
创建者　creators
指派　assigning 7-33至7-34，7-47至7-49
定义　defined 7-9至7-10
磁盘，打开磁盘插入事件　disks，opening 7-20
磁盘插入事件　disk-inserted events
示警框和对话框中的　in alert and dialog boxes 6-87
处理　handling 2-55至2-56
和 ModalDialog过程　and the ModalDialog procedure 2-56
和标准文件对话框　and standard file dialog boxes 2-55
次切换　minor switches 2-16
从Finder启动应用程序　launching applications，from the Finder 7-25至7-26
从操作系统事件队列中删除事件　flushing events from 2-93

D

打开　opening
打开应用程序，从Finder 中——应用程序 applications，from the Finder 7-25至7-26
打开磁盘　disks 7-20
打开文稿　documents 7-25至7-26
打开文件夹　folders 7-20
“打开”命令（文件菜单）　Open command　(File menu)　3-23
打开文稿事件　Open Documents events 7-20，7-25至7-26
打印监控程序文稿文件夹　PrintMonitor Documents folder
的替身类型　alias type for 7-40
定义　defined 7-44
查找　finding 7-54
“打印”命令（文件菜单）　Print command　(File menu)　3-23
打印文稿事件　Print Documents events 7-25至7-26

大 4- 位彩色图标资源 large 4-bit color icon resources
创建 creating 7-11，7-24
的Rez 输出格式 Rez output format for 7-59至 7-60
大 8 位彩色图标资源 large 8-bit color icon resources
创建 creating 7-11，7-24
的Rez 输出格式 Rez output format for 7-61至 7-62
大（32X32 像素）图标 large (32-by-32 pixel) icons 7-10，7-11
大小菜单 Size menu 3-27至3-29，3-82
处理 handling 3-82至3-84
中的“其它”命令 Other command in 3-28至 3-29
单选按钮 radio buttons
有效 active 5-11至5-13
的控制定义标识符 control definition ID for 5-14
创建，在窗口中创建 creating，in windows 5-20 至5-21
定义 defined 5-6
中的事件 events in 6-78
窗口的…中的事件 in windows 5-31至5-32
无效 inactive 5-11至5-13
的部位代码 part code for 5-32
指定示警框和对话框中的 specifying in alert or dialog boxes 6-153
当前菜单列表 current menu list 3-40
当前过程 current process 2-28
“当心”示警框 caution alerts
用CautionAlert 函数创建 creating with the CautionAlert function 6-111
定义的 defined 6-9
等效命令键 Command-key equivalents。参见等效组合键
等效组合键 keyboard equivalents 3-18至3-19
通常使用的 commonly used 3-19
定义 defining 3-47
保留的 reserved 3-18
低级事件 low-level events
处理 handling 2-32至2-57
接收 receiving 2-21
定义“注意”示警框 defined 6-8
定义操作系统事件 defined 2-10至2-12
定义操作系统事件队列 defined 2-6
定义的窗口定义标识符 defined 4-11
定义空事件 defined 2-9
定制控制部件 custom controls 5-11，5-109至5-115
定制图标 customized icons 7-17至7-18
“订户选项”命令（编辑菜单） Subscriber Options command (Edit menu) 3-25
动作过程 action procedures
关于按钮、复选框和单选按钮 for buttons，checkboxes，and radio buttons 5-36
更改 changing 5-102
创建 creating 5-115至5-117
定义 defined 5-35
关于弹出式菜单 for pop-up menus 5-36至5-37
关于滚动条 for scroll bars 5-58至 5-61
说明 specifying 5-91，5-92
端口矩形 port rectangle 4-18，4-101，4-103
端口名称 port names 2-75
转换成进程序号 converting至process serial numbers 2-105
和接收高级事件 and receiving high-level events 2-73
堆栈，堆栈的大小 stack，size of 2-32
对话管理器 Dialog Manager 6-5至6-172
应用程序为…定义的例程 application-defined routines for 6-143至6-147
控制管理器 and Control Manager 5-3，5-5，5-16，5-29，5-31，6-78
中的数据结构 data structures in 6-101 至6-102
初始化 initializing 6-102至6-105
本地化准则 localization guidelines for 6-34至6-35，6-39，6-40，6-80，6-130
和菜单管理器 and Menu Manager 6-68至6-73
中的资源 resources in 6-147至6-164。参见警告颜色表资源；警告资源；对话颜色表资源；对话资源；项目颜色表资源；项目列表资源
的例程 routines in 6-102至6-147
测试AppendDITL、ShortenDITL和CountDITL的可用性 testing for availability of AppendDITL，ShortenDITL，and CountDITL routines 6-18
和 and TextEdit 6-79至6-80
用户界面准则 user interface guidelines for 6-8至6-12，6-31至6-34，6-37至6-41，6-42，6-62至6-64
和窗口管理器 and Window Manager 4-

13至4-14，6-15至6-16
对话记录　dialog records 6-101至6-102
对话框　dialog boxes
存取…的菜单　accessing menus from 3-84，6-68至6-73
示警框和…的区别　alert boxes, as distinguished from 6-6至6-8
关闭　closing 6-100至6-101
创建　creating 6-23至6-26，6-113至6-119
改变…的默认颜色　default colors of, changing 6-156至6-157
定义　defined 6-5
显示　displaying 6-61至6-75
处理　disposing of 6-113，6-119至6-120
的事件过滤函数　event filter function for 6-86至6-89
中的事件　events in 2-25，2-29至2-30，6-82至6-100，6-135至6-143
的实例　examples of 1-9
中的项目　items in。参见示警框和对话框中的项目
的类型　types of 6-9至6-12
作为窗口　as windows 6-15至6-17
的窗口类型　window types for 4-9至4-10
对话框的窗口定义标识符　for a dialog box 6-25
对话项目参考号　session reference number 2-73
对话颜色表资源　dialog color table resources 6-75，6-156至6-157
对话资源　dialog resources
定义　defined 6-15
的实例　example of 6-24
的Rez 输入格式　Rez input format for 6-24至6-26
的Rez 输出格式　Rez output format for 6-148至6-149

F

非当前活动窗口　inactive windows 4-6至4-7
处理…中的鼠标按下事件　handling mouse down events in 4-43
非当前活动窗口中的"首次单击"　"first click" in an inactive window 4-7，4-42
废纸篓及其的替身类型　Trash, alias type for 7-40
废纸篓目录　Trash directories
关于…的外观　appearance至users 7-42
定义　defined 7-44
查找　finding 7-54
分隔线　dividers 3-12
分配存储器　memory, allocating 1-15
分区及其大小　partitions, sizes of 2-30，2-115
服务器及其替身类型　servers, alias type for 7-40
辅助窗口记录　auxiliary window record 4-21，4-73至4-74
辅助控制记录　auxiliary control records 5-76至5-77，5-107
复选框　checkboxes
激活的　active 5-11至5-13
改变设定和显示　changing setting and display of 5-38至5-39
的控制部件定义标识符　control definition ID for 5-14
定义的　defined 5-5至5-6
中的事件　events in 6-78
窗口中的　in windows 5-31至5-32，5-38至5-39
不活动的　inactive 5-11至5-13
的部位代码　part code for 5-32
在示警框或对话框中说明　specifying in alert or dialog boxes 6-29至6-30，6-153

G

改变菜单项目的语系　script, changing a menu item's 3-62，3-154
高级事件　high-level events。参见苹果事件 2-13至2-15，2-67至2-78
的接受　accepting 2-69，2-70
定义　defined 2-13
确定…的发送程序　determining the sender of 2-72
的事件类　event classes 2-68
的事件标识符　event IDs 2-68
处理　handling 2-67至2-78
投寄选项　posting options 2-75
接收　receiving 2-68，2-87
应答　replying to 2-74
和回执　and return receipts 2-77
在…中搜索指定的事件　searching for a specific event 2-71至2-72，2-92

发送　sending 2-73至2-77，2-101
支持　supporting 2-25
测试…的可用性　testing for availability 2-7
高级事件队列　high-level event queue 2-7，2-13
高级事件消息记录　high-level event message record 2-72，2-82至2-83
更新区域　update region
定义　defined 4-13
保留　maintaining 4-41
重新绘制　redrawing 4-40至4-41
更新事件　update events 4-13
示警框中的　in alert boxes 6-82，6-86至6-89
定义　defined 2-9
对话框中的　in dialog boxes 6-85，6-86至6-89，6-97至6-100
处理　handling 2-47至2-50，4-48至4-50
处理…的例程　routines for handling 4-106至4-107
窗口的　for windows 4-41
工具箱事件管理器　Toolbox Event Manager。参见事件管理器
功能扩展文件夹　Extensions folder
的替身类型　alias type for 7-40
定义的　defined 7-44
查找　finding 7-54
'extn'文件夹类型　'extn' folder type 7-54
共享文件夹及其替身类型　shared folder, alias type for 7-40
固定样板文稿　stationery pads
创建…的图标　creating icons for 7-16，7-19
的默认图标　default icon for 7-12
定义　defined 7-4，7-34至7-36
分辨　recognition of 2-119
挂起事件　suspend events
定义　defined 2-10
处理　handling 2-60至2-62
"关闭"命令（文件菜单）　Close command (File menu) 3-23
关闭窗口　closing windows 4-60至4-62
的例程　routines for 4-103至4-106
关闭框　close box 4-6
关闭区域　close region 4-12
关于命令（苹果菜单）　About command (Apple menu) 3-21至3-22，3-45
光标　cursors
调整…形状　adjusting 2-65至2-67
箭头　arrow 2-63
十字　cross 2-63
I型　I-beam 2-63
加号　plus 2-63
设定…形状　setting the appearance of 2-62
手表　wristwatch 2-63
规模资源　size resources 2-115至2-119，7-35
滚动箭头　scroll arrows
的动作过程　action procedures for 5-58至5-61
定义的　defined 5-8至5-9
中的事件　events in 5-57至5-61
的部位代码　part codes for 5-32
滚动框　scroll boxes
的动作过程　action procedure for 5-36
定义　defined 5-7至5-8
拖拽　dragging 5-35
中的事件　events in 5-53至5-57
移动　moving 5-9至5-10，5-39至5-43，5-44，5-61，5-70
的部位代码　part code for 5-32
滚动条中的箭头　arrows, in scroll bars。参见滚动箭头
滚卷条　scroll bars。参见灰度区域；滚动箭头；滚动框
激活　activating 5-13
有效　active 5-11至5-13
改变…的设定值和显示　changing settings and display of 5-9至5-10，5-39至5-43，5-44，5-61，5-70
的部件标识符　control definition ID for 5-14
创建…，在窗口中创建　creating, in windows 5-21至5-25
定义　defined 5-7至5-10
用…滚动文稿　document scrolling with 5-9至5-10，5-43至5-70
中的事件　events in 5-31至5-32，5-43至5-65
无效　inactive 5-11至5-13
和列表管理器　and List Manager 5-3
移动…及调整…的大小　moving and resizing 5-65至5-70
更新　updating 5-30
和窗口元素　as a window element 4-6
滚卷条中的灰度区域　gray areas, in scroll bars
对于…的动作过程　action procedures for 5-58至5-61
定义　defined 5-9

的事件 events in 5-57至5-61
的部位代码 part codes for 5-32
滚卷条中的页面区域 page regions, in scroll bars 。参见灰度区域
过滤函数 filter function。参见事件过滤函数
GetSpecificHighLevelEvent 的… for GetSpecificHighLevelEvent 2-92至2-93, 2-114至2-115

H

后台进程 background processes 2-4
后台应用程序，及功能扩展文件夹 background applications, and Extensions folder 7-44
滑块控制功能 slider controls 5-11
灰度区域 gray region 4-16
恢复事件 resume events
定义 defined 2-10
处理 handling 2-60至2-62
回执 return receipts 2-75, 2-77
活动窗口 active windows 4-6至4-7
活动控制部件 active controls 5-11至5-13, 6-13至6-14

J

激活事件 activate events
在示警框中 in alert boxes 6-82
失活示警框和对话框 deactivating windows for alert and dialog boxes 6-64至6-66
定义 defined 2-8
在对话框中 in dialog boxes 6-85, 6-97至6-100
处理 handling 2-50至2-55, 4-50至4-53
和窗口 and windows 4-42
“剪”命令（编辑菜单） Cut command (Edit menu) 3-25, 6-69
剪贴板 Clipboard 3-24
转换数据自和到 converting data至and from 2-58至2-60
“隐藏剪贴板”命令（编辑菜单） Hide Clipboard command (Edit menu) 3-25
“显示剪贴板”命令（编辑命令） Show Clipboard command (Edit menu) 3-25
剪贴区 scrap
全局剪贴区到私有剪贴区的转换 converting global至private 2-12
私有剪贴区到全局剪贴区的转换 converting private至global 2-12
剪贴区管理器 Scrap Manager 1-14, 2-60
键按下事件 key-down events
定义 defined 2-8
处理…的等效组合键 handling keyboard equivalents 3-77至3-78
接收 receiving 2-38至2-46
键盘 keyboards 2-40
苹果扩展键盘 II,美国版的键盘 Apple Extended Keyboard II, domestic 2-43
苹果扩展键盘 II,ISO 版的键盘 Apple Extended Keyboard II, ISO 2-43
苹果键盘 II,ISO 版的键盘 Apple Keyboard II, ISO 2-42
获得键盘 状态 getting the state of 2-110
键盘事件 keyboard events
示警框中的 in alert boxes 6-86至6-89
对话框中的 in dialog boxes 6-79至6-80, 6-86至6-89, 6-94至6-97
处理 handling 2-38至2-45
窗口中的 in windows 4-47
键盘布局资源 keyboard layout resources 2-39至2-42, 3-17, 3-32, 3-33
键盘菜单 Keyboard menu 3-32至3-33
从示警框和对话框中存取 accessing from alert and dialog boxes 6-68至6-69
菜单管理器添加的 added by Menu Manager 3-19
键盘资源 keyboard resources
的图标 icon for 7-37至7-38
‘KCHR’键盘资源 'KCHR' 2-39至2-42
‘KMAP’键盘资源 'KMAP' 2-21, 2-39至2-42
键释放事件 key-up events 2-8, 2-29
箭头光标 arrow cursor 2-63
结构区域 structure region 4-6, 4-12
进程 processes
后台进程 background 2-4
进程的内容 context of 2-15
进程管理器 Process Manager
和事件管理器 and Event Manager 2-3
和 and the Finder 7-25
进程内容，进程内容的切换 context of a process, switching 2-15至2-16
进程序号转换成端口名称 process serial numbers,

converting至port names 2-105至2-107
警告 alerts
创建 creating 6-18至6-23，6-105至6-112
定义的 defined 6-5
定义示警声 defining alert sounds for 6-22，6-144至6-145
显示 displaying 6-61至6-75
中的事件 events in 6-81至6-82
的阶段 stages for 6-20至6-22，6-150至6-151
的类型 types of 6-8至6-9
使用系统示警声的 using the system alert sound 6-8
警告颜色表资源 alert color table resources 6-157至6-158
警告资源 alert resources
定义 defined 6-15
的实例 example of 6-20，6-21
的Rez 输入格式 Rez input format for 6-19至6-22
的Rez 输出格式 Rez output format for 6-150至6-151
静态文本项目 static text items
从…中获取文字 getting text from 6-130至6-131
在…中设定文字 setting text in 6-36至6-41，6-131，6-153
指定示警框和对话框中的 specifying in alert or dialog boxes 6-29至6-30，6-153
替换…中的文字 substituting text in 6-46至6-48
替换…中的文字字符串 substituting text strings in 6-129至6-130
的用户界面准则 user interface guidelines for 6-40至6-41
静态文字，示警框和对话框中的静态文字 static，in alert and dialog boxes。参见静态文本项目
局部 ID local IDs
束资源和文件参考资源中的 in bundle resources and file reference resources 7-19，7-22，7-24
映射资源的 for mapping resources 7-18至7-25
本地化准则 localization guidelines
控制管理器的 for Control Manager 5-15，5-19至5-20，5-85，5-120
对话管理器的 for Dialog Manager 6-34至6-35，6-39，6-40，6-80，6-130
位置名称 location names 2-73，2-75
局部坐标 local coordinates 4-18
矩形，控制部件的窗口类型5-18至5-19，5-25至5-26，5-82，5-118。亦请参 rectangles，for controls
卷，Finder 的卷编目桌面数据库 volumes，Finder's desktop database for 7-45
卷编目，卷编目中的Finder 信息 volume catalogs，Finder information in 7-32至7-34

K

“拷贝”命令（编辑菜单） Copy command (Edit menu) 3-25，6-69
可编辑文字，对话框中的可编辑文字 editable，in dialog boxes。参见可编辑文字项目
可编辑文字项目 editable text items
中的事件 events in 6-79至6-80
从…中获取文字 getting text from 6-48至6-51，6-130至6-131
醒目显示…中的默认文字 highlighting default text in 6-131至6-132
在…中设定文字 setting text in 6-131，6-153
指定对话框中的 specifying in dialog boxes 6-29至6-30，6-153
可视区域 visible region 4-48
可移动模式对话框 movable modal dialog boxes
从…中存取菜单 accessing menus from 6-68至6-73
关闭 closing 6-100至6-101
创建 creating 6-23至6-26，6-113至6-119
定义 defined 6-11至6-12
显示 displaying 6-61，6-62
…中的事件 events in 6-89至6-100，6-138至6-141
…的标题 titles for 6-25
…的窗口类型 window types for 4-9
可移动资源 movable resources 7-36至7-38
空事件 null events
空事件和事件屏蔽参数 and event masking 2-28
控制板，控制板的安装和删除 control panels，installing and removing 7-41
控制板文件夹 Control Panels folder
的替身类型 alias type for 7-40
定义 defined 7-43
查找 finding 7-54
控制部件 controls
的动作过程 action procedures for 5-91，

5-102，5-115至5-117
激活 activating 5-13
活动 active 5-11至5-13，6-13
示警框和对话框中的 in alert and dialog boxes 6-29至6-30，6-154
的色彩 colors in 5-5，5-101，5-121至5-123
的控制定义标识符 control definition IDs for 5-18，5-19，5-25，5-83至5-84，5-120
的定制 custom。参见定制的控制功能
定义 defined 5-3
拖拽 dragging 5-35，5-99至5-100，5-114至5-115
绘制 drawing 5-17，5-85至5-88
中的事件 events in 5-88至5-93，6-78
的醒目显示状态 highlight states of 5-35，5-98至5-99
无效 inactive 5-13，6-13，6-74
中的指示符 indicators in。参见指示符
在不可见…中 invisible 5-18，5-19，5-25，5-83，5-97，5-119
中的鼠标事件 mouse events in 5-11至5-13，5-30至5-70
移动 moving 5-65至5-70，5-97至5-98
的部位代码 part codes for 5-31至5-32
弹出式菜单的 pop-up menus 6-42至6-44
的矩形 rectangles for 5-18至5-19，5-22至5-23，5-25至5-26，5-82，5-118。参见显示矩形
的参考值 reference values for
改变…的参考值 changing 5-105至5-106
确定…的参考值 determining 5-105
指定…的参考值 specifying 5-18，5-25，5-84，5-120
删除 removing 5-108至5-109
调整的…大小 resizing 5-65至5-70，5-98
的资源 resources for。参见控制功能资源
的设定状态 settings of
改变…的设定状态 changing 5-11，5-37至5-43，5-61，5-65至5-70，5-93至5-102
确定…的设定状态 determining 5-37至5-43，5-102至5-107
设定…的初始设定状态 specifying initial 5-18至5-19，5-21至5-23，5-25至5-28，5-83，5-118至5-119
标准 standard 5-4，5-15至5-28，5-81至5-85
中的文字 text in 5-18，5-19至5-20，5-25
的标题 titles of
改变…的标题 changing 5-96
确定…的标题 determining 5-104至5-105
设定…的标题 specifying 5-18，5-19至5-20，5-25，5-83，5-85，5-120，5-121
的类型 5-4至5-13 types of 5-4至5-13。参见按钮；核查框；自定控制功能；弹出式菜单；单选按钮；滚动条
更新 updating 5-29至5-30
的可见性 visible 5-18，5-19，5-25，5-83，5-119
窗口中的 in windows 4-14至4-15，4-23，5-82
控制部件定义标识符 control definition IDs
按钮的 for buttons 5-14
复选框的 for checkboxes 5-14
自定义 for custom controls 5-109至5-110
定义 defined 5-14
弹出式菜单的 for pop-up menus 5-14
单选按钮的 for radio buttons 5-14
滚动条的 for scroll bars 5-14
设定 specifying 5-18，5-19，5-25，5-83至5-84，5-120
控制部件定义函数 control definition functions
自定义控制部件 custom 5-109至5-115
定义 defined 5-14
弹出式菜单的 for pop-up menus 5-14，5-36
标准 standard 5-14
控制部件列表 control list 4-14
控制部件中指示符 indicators，in controls
定义 defined 5-11
拖拽 dragging 5-35，5-99至5-100，5-114至5-115
控制管理器 Control Manager 5-3至5-135
应用程序为定义的例程 application-defined routines for 5-109至5-117
中的数据结构 data structures in 5-72至5-80
和对话管理器 and Dialog Manager 5-3，5-5，5-16，5-29，5-31，6-78
和事件管理器 and Event Manager 5-3
和列表管理器 and List Manager 5-3
本地化准则 localization guidelines 5-15，5-19至5-20，5-85，5-120

和资源管理器　and Resource Manager 5-3
的资源　resources for 5-117至5-123。参见控制颜色表；控制定义函数；控制资源；
的例程　routines in 5-80至5-117
用户界面准则　user interface guidelines 5-5至5-13，5-52至5-53
和窗口管理器　and Window Manager 5-3
控制记录　control records 5-73至5-75
控制色彩表　control color tables 6-160至6-162
控制色彩表记录　control color table records 5-77至5-80
控制色彩表资源　control color table resources 5-121至5-123
控制资源　control resources
的实例　example of 5-18，5-21，5-23，5-26
的Rez 输入格式　Rez input format for 5-18至5-24，5-25至5-28
的Rez 输出格式　Rez output format for 5-118至5-121
框架，窗口——　frames，window 4-6，4-12至4-13
扩展的目录信息记录　extended directory information record 7-50至7-51
扩展的文件信息记录　extended file information record 7-49

L

来自卷名文件夹的拯救项目　Rescued Items from volume name folders 7-44
列表　listings。参见示例例程
列表管理器和控制管理器　List Manager，and Control Manager 5-3
临时文件　temporary files 7-43，7-44
临时项目文件夹　Temporary Items folder
定义　defined 7-43
查找　finding 7-54

M，N

面向用户的设计　user-oriented design 1-5
模拟键代码　virtual key codes 2-40
模式对话框　modal dialog boxes
从…中存取菜单　accessing menus from 6-68至6-73
关闭　closing 6-100至6-101
创建　creating 6-23至6-26，6-113至6-119
定义　defined 6-10至6-11
显示　displaying 6-61，6-62至6-75
的事件过滤函数　event filter function for 6-86至6-89，6-145至6-147
中的事件　events in 6-82至6-86，6-135至6-137
的窗口类型　window types for 4-9
默认按钮　default buttons 6-10至6-11，6-22，6-30至6-31，6-56至6-61
拇指型控制部件　thumb controls。参见滚动框
目标标识符记录　target ID records 2-81至2-82
接收　receiving 2-73
发送　sending 2-75
目录文件　catalog files
定义　defined 7-5
中的Finder信息　Finder information in 7-32至7-34，7-46至7-51
目录信息记录　directory information record 7-50
内容区域　content region
定义　defined 4-6，4-12
绘制　drawing 4-39至4-40
更新　updating 4-40至4-41，4-48至4-50

P

旁述　help balloons
图标的　for icons 7-38
窗口框架的　for window frames 4-23
旁述联机帮助，旁述图标　Balloon Help online assistance，for icons 7-38。参见旁述
旁述资源　help balloon resources 3-44，3-68
苹果菜单　Apple menu 3-20
“关于”命令　About command 3-21至3-22，3-45
从对话框存取　accessing from dialog boxes 6-68至6-73
添加项目到　adding items to 3-68至3-69
创建　creating 3-43至3-44
处理　handling 3-80至3-81
苹果菜单项目文件夹　Apple Menu Items folder
从…向苹果菜单添加项目　adding items from，to Apple menu 3-20至3-21
的替身类型　alias type for 7-40
定义的　defined 7-43

查找 finding 7-54
苹果键盘 II，ISO 版 of Apple Keyboard II，ISO 2-42
苹果扩展键盘 II，ISO 版 of Apple Extended Keyboard II，ISO 2-43
苹果扩展键盘 II，美国版 of Apple Extended Keyboard II，domestic 2-43
苹果事件 Apple events
应用程序发给自己的 application sending to itself 3-71
定义 defined 2-13
和 and the Finder 7-25 至 7-26
处理 handling 2-78
作为高级事件 as high-level events 2-67 至 2-69
打开文稿 Open Documents 7-20
发送 sending 2-78
响应菜单命令 in response 至 menu commands 3-71
苹果事件进程间通信协议 Apple Event Interprocess Messaging Protocol (AEIMP) 2-13，2-67

Q，R

“其它”命令（大小菜单） Other command (Size menu) 3-28 至 3-29，3-82 至 3-84
启动项目文件夹 Startup Items folder
的替身类型 alias type for 7-40
定义 defined 7-44
查找 finding 7-54
启用项目 enabled items 6-13
前台进程 foreground 2-4
进程间的切换 switching between 2-5，2-15
前台进程 foreground process 2-4
嵌套式菜单 hierarchical menus 3-38，3-53 至 3-56
强制剪贴区 scrap coercion 2-58，2-61
切换 switching
切换内容 context 2-15
主切换 major 2-16
次切换 minor 2-16
“清除”命令（编辑菜单） Clear command (Edit menu) 3-25，6-69
“取消”命令（编辑菜单） Undo command (Edit menu) 3-25，6-69
取消事件 cancel events 2-46 至 2-47
全局坐标 global coordinates 4-17
“全选”（编辑菜单） Select All command (Edit menu) 3-25
人机界面准则 human interface guidelines。参见用户界面准则
软盘，...的替身类型 floppy disks，alias type for 7-40

S

色彩 colors
示警框和对话框中的 in alert boxes and dialog boxes 6-156 至 6-164
控制部件的 in controls 5-5
标签菜单命令 Label menu commands for 7-16
菜单中的 in menus 3-46，3-62，3-98 至 3-102
窗口中的 in windows 4-20 至 4-21
示警框 alert boxes
存取菜单 accessing menus from 6-68 至 6-73
创建 creating 6-18 至 6-23
系统默认颜色，改变 default colors of，changing 6-157 至 6-158
定义 defined 6-6
与对话框的差异 dialog boxes，as distinguished from 6-6 至 6-8
显示 displaying 6-22，6-61 至 6-75
的事件过滤函数 event filter function for 6-86 至 6-89，6-145 至 6-147
中的事件 events in 2-29 至 2-30，6-81 至 6-82
中的项目 items in。参见示警框和对话框中的项目
阶段 stages for 6-20 至 6-22，6-150 至 6-151
类型 types of 6-8 至 6-9
作为窗口 as windows 6-15 至 6-17
窗口类型 window types for 4-9
示警框和对话框中的图片 pictures，in alert or dialog boxes 6-29 至 6-30，6-154
示警框和对话框中的项目 items in alert and dialog boxes。参见定义应用程序的项目；按钮；核查框；控制功能；可编辑文字项目；项目列表资源，图片；弹出式菜单；单选按钮；静态文字项目
添加 adding 6-51 至 6-56，6-125 至 6-127

从…拷贝可编辑文字 copying editable text from 6-133
对…计数 counting 6-128
创建 creating 6-26至6-44
从…剪切可编辑文字 cutting editable text from 6-132至6-133
的默认颜色,改变…的默认颜色 default colors of，changing 6-158至6-164
定义 defined 6-13
从…中删除可编辑文字 deleting editable text from 6-134
为…显示热血 display rectangles for 6-29,6-32至6-35
绘制应用程序定义的 drawing application-defined 6-143至6-144
启用的和停用的 enabled and disabled 6-13，6-36
查找…的项目编号 finding item numbers of 6-125
获取…的间接指针 getting handles to 6-121至6-122
从…中获取文字字符串 getting text strings from 6-130至6-131
隐藏 hiding 6-123至6-124
醒目显示…中的默认文字 highlighting default text in 6-131至6-132
的项目编号 item numbers for 6-28
键盘引导的 keyboard navigation 6-44
操纵 manipulating 6-44至6-56，6-120至6-128
向…贴可编辑文字 pasting editable text into 6-133至6-134
的图片 pictures 6-29至6-30
隐藏…后重新显示 redisplaying after hiding 6-124
删除 removing 6-127至6-128
设定或改变 setting or changing 6-122至6-123
在…中设定文字字符串 setting text strings in 6-131
替换…中的文字 substituting text in 6-129至6-130
的类型 types of 6-13至6-14，6-30
示警声 alert sounds
创建 creating 6-18至6-22
定义的 defined 6-6
定义 defining 6-22，6-144至6-145
说明 specifying 6-151
示警声的阶段 stages for 6-20至6-22，6-150至6-151
和系统示警声 and the system alert sound 6-8
示例代码 sample code。参见示例例程
示例例程 sample routines
DisplayMyDialog DisplayMyDialog 6-58
DoActivate 2-53，4-51至4-52
DoActivateGlobalChangesDialog 6-98
DoCloseCmd 4-60至4-61，6-94
DoContentClick 5-33，5-53，6-92
DoCopyResource 7-29
DoDiskEven DoDiskEvent 2-56
DoDrumRollCheckBox 5-38
DoEvent DoEvent 2-26，3-72，3-77
DoGlobalChangesDialog DoGlobalChangesDialog 6-67
DoGraphicsScroll DoGraphicsScroll 5-47
DoGrowWindow 4-58至4-59
DoHighLevelEvent 2-70
DoIdle 2-57，6-79
DoKeyDown DoKeyDown 2-44，3-78，6-95
DoMenuCommand DoMenuCommand 3-79
DoMouseDown 2-34，3-72，4-44至4-45，5-32，6-91
DoNew DoNew 5-24
DoNewCmd 4-28至4-29
DoOpenFile 4-37至4-38
DoOSEvent DoOSEvent 2-59
DoPlayButton 5-36
DoPopUpMenu DoPopUpMenu 5-37
DoSaveAsCmd DoSaveAsCmd 7-28
DoShowModelessFindDialogBox 4-64
DoSpellBoxWithSpanish 6-54
DoSuspendResumeEvent 2-61
DoUpdate 2-50，4-50，5-29，5-62，6-99
DoZoomWindow 4-55至4-56
IsStationeryDoc 7-36
MyAddHelpItem 3-68
MyAdjustCursor 2-65
MyAdjustEditMenuForDocWindow 3-75
MyAdjustEditMenuForModalDialog
MyAdjustEditMenuForModalDialogs 6-72
MyAdjustFileMenuForDocWindow 3-74
MyAdjustHV 5-41
MyAdjustMenu MyAdjustMenus 3-73，

6-70
MyAdjustMenuForDialog MyAdjustMenusForDialogs 6-70
MyAdjustScrollBars 5-39
MyAdjustScrollSizes 5-67
MyAdjustScrollValues 5-40
MyAlert 6-66
MyAlertSound 6-22
MyChang MyChangeMenuBarAndSaveColorInfo 3-52
MyCloseDocument 4-61至4-62，6-47
MyCreatePlaySoundsWindow 5-17
MyDrawDefaultButtonOutline 6-59
MyDrawGraphicsWindow 5-63
MyDrawWindow 4-39，5-30，5-65
MyEventFilter 6-88
MyEventLoop 2-24
MyGetLocalUpdateRgn 4-60
MyGetWindowType 4-24
MyHandleAppleCommand 3-80
MyHandleHelpCommand 3-81
MyHandleKeyDow MyHandleKeyDown 2-44，6-95
MyHandleKeyDownInModeless 6-96
MyHandleModelessDialogs 6-49
MyHandleSi MyHandleSizeCommand 3-83
MyHorzntlActionProc 5-59
MyMakeAllMenu MyMakeAllMenus 3-70
MyMakeMenus 3-50
MyMakeSun MyMakeSubMenu 3-55
MyMDEF 3-89
MyMoveScrollBox 5-61
MyOpen 7-41
MyPostTest 2-74
MyPostWithPPCBrowser 2-76
MyResizeWindow 4-59
MySaveWindowPosition 4-34至4-35
MySetWindowPosition 4-36至4-37
MySpellCheckDialog 6-83
MyToggleHideShow 3-59
MyVerticalActionProc 5-59
ShowMyAboutBox 6-65
UserDidCancel 2-46
“存储为”命令（文件菜单） Save As command (File menu) 3-23
“存储”命令（文件菜单） Save command (File menu) 3-23
SaveOld 过程 SaveOld procedure 4-117
SaveUpdate 全局变量 SaveUpdate global variable 4-118
SaveVisRgn 全局变量 SaveVisRgn global variable 4-106
事件 events
激活 activate 2-8，2-50至2-55，4-50至4-53
示警框中的 in alert boxes 2-29至2-30，6-81至6-82，6-86至6-89，6-135
苹果 Apple。参见苹果事件
自动键 auto-key 2-8，2-38
窗口按钮中的 in buttons in windows 5-31至5-37
取消 cancel 2-46至2-47
窗口复选框中的 in checkboxes in windows 5-38至5-39
“命令”＋“.”组合键事件 Command-period 2-46
控制部件中的 in controls 5-11至5-13，5-30至5-70，5-88至5-93，6-78
定义 defined 2-4
桌面附件中的 in desk accessories 2-36
对话框中的 in dialog boxes 2-29至2-30，6-77至6-100，6-135至6-143
磁盘插入 disk-inserted 2-8，2-55至2-56
可编辑文字项目中的 in editable text items 6-79至6-80
示警框和对话框中的…过滤函数 filter function for，in alert and dialog boxes 6-86至6-89，6-145至6-147
非当前活动窗口中首次单击 first click in an inactive window 4-42
获得有关…的信息 getting information about 2-18至2-21
滚动条灰度区域中的 in gray areas of scroll bars 5-57至5-61
处理 handling 2-26
高级 high-level。参见高级事件
从后台和用户交互作用的 interacting with the user from the background 2-12
键盘 keyboard 2-8，2-38至2-45
键按下 key-down 2-38
键释放 key-up 2-38，2-39
低级 low-level 2-8至2-9
屏蔽 masking out 2-27
模式对话框中的 in modal dialog boxes 6-82至6-89，6-135至6-137

无模式对话框中的　in modeless dialog boxes 2-25，6-89至6-100，6-138至6-141
和修改键　and modifier keys 2-20至2-21，2-34
按下鼠标 2-8，2-33至2-8　mouse-down 2-8，2-33至2-38
窗口中的按下鼠标　mouse-down in windows 4-42至4-45
移动鼠标　mouse-moved 2-12，2-23，2-62至2-67
释放鼠标　mouse-up 2-8，2-33
可移动模式对话框中的　in movable modal dialog boxes 6-89至6-100，6-138至6-141
空　null 2-9，2-57
操作系统　operating-system 2-10至2-12，2-58至2-67
弹出式菜单中的　in pop-up menus 6-42
窗口的弹出式菜单中的　in windows 5-31至5-37
优先级　priority of 2-15
实例　processing 2-21至2-30
窗口的单选按钮中的　in radio buttons in windows 5-31至5-32
在后台收到的　received in the background 2-12
从其它应用程序收到的　receiving from otherapplications 2-69至2-71
继续执行　resume 2-12，2-60至2-62
滚动箭头中的　in scroll arrows 5-57至5-61
滚卷条中的　in scroll bars 5-43至5-65
滚动框中的　in scroll boxes 5-53至5-57
源　sources of 2-6
暂停　suspend 2-12，2-60至2-62
交换内容的　switching contexts 2-15至2-16
类型　types of 2-5，2-18，2-79
更新　update 2-9，2-47至2-50，4-48至4-50
窗口中的　in windows 4-21至4-22，4-41至4-53
类型　event types 2-5，2-18，2-79
事件标识符　event IDs 2-68
事件队列　event queue。参见高级事件队列；操作系统事件队列
扫描…中的取消事件　scanning for a cancel event in 2-46
的结构　structure of 2-83
事件管理器 2-3至2-132　Event Manager。参见操作系统事件管理器
应用程序为…定义的例程　application-defined routine for 2-114至2-115
和控制管理器　and Control Manager 5-3
中的数据结构　data structures in 2-79至2-83
初始化　initializing2-17
和MultiFinder　and MultiFinder 2-7
和 PPC 工具箱　and the PPC Toolbox 2-7，2-13
和进程管理器　and the Process Manager 2-5
的例程　routines in 2-84至2-113
和剪贴区管理器　and the Scrap Manager 2-60
功能测试　testing for features 2-17
事件过滤函数　event filter functions 6-86至6-89，6-145至6-147
事件记录　event records 2-79至2-81
定义　defined 2-4
和事件循环　and event loops 2-25
高级事件的　for high-level events 2-68，2-69
事件类　event classes 2-68
事件屏蔽参数　event masks
定义　defined 2-22，2-85
设定　setting 2-26至2-29
事件消息　event messages 2-19，2-80
事件循环 2-24至2-25　event loops
释放鼠标事件　mouse-up events 2-33
示警框中的　in alert boxes 6-78，6-81至6-82
控制部件中的　in controls 5-11至5-13，5-30至5-70
对话框中的　in dialog boxes 6-78至6-80，6-82至6-94
手表光标　wristwatch cursor 2-63
手提式字形和桌面附件　suitcases for fonts and desk accessories 7-37
输出的 AppleShare 文件夹，…的替身类型　exported AppleShare folder，alias type for 7-40
输入方法　input methods 3-32
署名　signatures
束资源中的署名　in bundle resources 7-20
作为创建者的署名　as creators 7-9
定义署名　defined 7-8
登记署名　registering 7-8至7-10
署名资源　signature resources 7-8至7-10
鼠标　mouse
确定鼠标的位置　determining location of 2-108
获取鼠标的信息　getting information about 2-108

鼠标按下事件 mouse-down events
示警框中的 in alert boxes 6-78，6-81至6-82
控制部件中的 in controls 5-11至5-13，5-30至5-70
桌面附件中的 in desk accessories 2-87，2-95
对话框中的 in dialog boxes 6-78至6-80，6-82至6-94
处理 handling 2-33至2-38
菜单条中的 in the menu bar 3-72至3-73
窗口中的 in windows 4-42至4-45
鼠标区域 mouse region 2-23，2-64至2-67
鼠标驱动程序 mouse driver 2-62
数据分支 data forks 1-11
双击，双击的时间间隔 double clicks，time between 2-38，2-113
睡眠值 sleep value 2-22
缩放窗口 zooming windows 4-53至4-56，4-101至4-103
缩放框 zoom box 4-6
缩放区域 zoom region 4-12

T

弹出式菜单 pop-up menus 3-33至3-38
的动作过程 action procedure for 5-36至5-37
启用的 active 5-11至5-13
的控制定义函数 control definition function for 3-33，3-34至3-36，5-14
的控制定义标识符 control definition ID for 5-14
创建 creating 3-56至3-57，5-25至5-28
定义 defined 3-6，5-6至5-7
对话框中的 in dialog boxes 6-42至6-44
中的事件，窗口的…中的事件 events in，in windows 5-31至5-37
停用 inactive 5-11至5-13
测试…的可用性 testing for availability 3-42
的键入域 type-in fields 3-37
的变量代码 variation codes for 5-27至5-28
调色板管理器和系统颜色表 Palette Manager，and system color tables 4-21
调整窗口大小 resizing windows 4-57至4-59
调整大小格 size box 4-6
调整区域 size region 4-12
替身 aliases
定义 defined 7-39
还原 resolving 7-39至7-41，7-51至7-53
替身记录 alias records 7-40
替身类型 alias types 7-40
替身文件 alias files 7-39
“贴”命令（编辑菜单） Paste command (Edit menu)
和对话框 and dialog boxes 6-69
和信息窗口 and information windows 7-17
引入的 introduced 3-25
“停止”示警框 stop alerts
用 StopAlert 函数创建 creating with the StopAlert function 6-109
定义的 defined 6-9
通知管理器和操作系统事件 Notification Manager，and operating-system events 2-12
投寄选项 posting options 2-75
“退出”命令（文件菜单） Quit command (File menu) 3-23
图标 icons。参见图标库
应用程序的 for applications 7-10至7-15
黑白 black and white 7-13，7-15
彩色 color 7-15至7-16
创建 creating 7-11至7-18
自定义 customizing 7-17至7-18
Finder使用的默认 defaults used by Finder 7-12，7-40
定义 defined 7-3
文稿的 for documents 7-16至7-18
出版数据的 for editions 7-16
8 位彩色 8-bit color 7-11
字体资源的 for font resources 7-37
4 位彩色 4-bit color 7-11
键盘布局资源的 for keyboard layout resources 7-37
大图标 large 7-10，7-13
菜单项目的 for menu items 3-62至3-64
菜单中的 in menus 3-12，3-62至3-64，3-154
询问文稿的 for query documents 7-16
缩小的 reduced 3-62至3-63
语系系统资源集合的 for script system resource collections 7-37
小图标 small 7-13，7-15
语音资源的 for sound resources 7-37

示警框和对话框中指定的　specifying in alert or dialog boxes 6-29至6-30，6-154
固定样板文稿的　for stationery pads 7-16,7-19,7-36
系统　system 7-12
TrueType 字体资源　for TrueType font resources 7-37
图标库　icon families 7-11
图标列表资源　icon list resources
创建　creating 7-11至7-15
图标　example of 7-14至7-15
的Rez 输出格式　Rez output format for 7-57至7-58
在束资源中指定　specifying in bundle resources 7-20
图标掩码　icon masks 7-13，7-15
图标资源（'ICON'资源类型）　icon resources ('ICON' resource type)　7-63
图形端口　graphics ports 4-17至4-18
拖拽区域　drag region 4-12

W

网络，在网络上使用共享的废纸篓目录　networks, using shared Trash directories on 7-44
未找到应用程序名字字符串资源　missing-application name string resources 7-27至7-30，7-68
文本样式表　text style table 6-160，6-162至6-164
文稿　documents 5-9至5-10，5-43至5-70
创建…的图标　creating icons for 7-16至7-17，7-17至7-18
的创建者　creators of 7-9至7-10
的默认图标　default icon for 7-12
的文件类型　file types of 7-9至7-10，7-18至7-20
从Finder打开　opening from the Finder 7-25至7-26
禁止用户从Finder打开　prohibiting users from opening from Finder 7-30
的窗口类型　window types for 4-8
文稿窗口　document windows
定义　defined 4-4
的定位　positioning 4-30
存储…的位置　saving position of 4-34
的窗口类型　window types for 4-8
文件　files。参见应用程序、文稿
数据分支　data fork of 1-11
管理　managing 1-15
资源分支　resource fork of 1-11
临时　temporary 7-43
和…有关的用户评论　user comments associated with 7-45
文件菜单　File menu 3-22至3-24
的“关闭”命令　Close command 3-23，7-18
的“新文稿”命令　New command 3-23
的“打开”命令　Open command 3-23
的“页面设置”命令　Page Setup command 3-23
的“打印”命令　Print command 3-23
的“退出”命令　Quit command 3-23，7-18
的“存储为”命令　Save As command 3-23
的“存储”命令　Save command 3-23
文件参考资源　file reference resources 7-18至7-25，7-64至7-65
文件夹　folders
的替身类型　alias type for 7-40
查找和系统有关的　finding system-related 7-53至7-56
文件夹类型　folder types 7-54
文件夹资源　folder resources 7-55
文件类型　file types
替身文件的　for alias files 7-40
'appe'文件类型　'appe'　7-33，7-47
'APPL'文件类型　'APPL'　7-10
的指派　assigning 7-9至7-10，7-33至7-34，7-47至7-49
'****'文件类型　'****'　7-20
在文件参考资源中定义的　defining in file reference resources 7-18至7-25
'DFIL'文件类型　'DFIL'　7-37
'disk'文件类型　'disk'　7-20
'FFIL'文件类型　'FFIL'　7-37
'ffil'文件类型　'ffil'　7-37
'fold'文件类型　'fold'　7-20
'ifil'文件类型　'ifil'　7-37
'kfil'文件类型　'kfil'　7-37
可移动资源的　for movable resources 7-37
打开特定…的文稿　opening documents of particular
从应用程序打开特定…的文稿　from applications 7-10

从 Finder 打开特定…的文稿 from the Finder 7-25至7-26
'????' 文件类型 '????' 7-20
'scri'文件类型 'scri' 7-44
'sfil'文件类型 'sfil' 7-37
固定样板文稿的 for stationery pads 7-19
'tfil' 文件类型 'tfil' 7-37
文件信息记录 file information record 7-34, 7-47至7-49
文字 text
按钮、复选框和单选按钮中的文字 in buttons, checkboxes, and radio buttons 5-18, 5-19, 5-19至5-20, 5-25, 6-37至6-40, 6-153
无模式对话框 modeless dialog boxes
从…中存取菜单 accessing menus from 6-68至6-73
关闭 closing 6-100至6-101
创建 creating 6-23至6-26, 6-113至6-119
定义 defined 6-12
显示 displaying 6-61至6-67
中的事件 events in 6-89至6-100, 6-138至6-141
的标题 titles for 6-25
的窗口类型 window types for 4-10
无效控制部件 inactive controls 5-13, 6-13至6-14
无效项目 disabled items 6-13

X

系统处理的菜单 system-handled menus 3-19
系统功能扩展 system extensions
和功能扩展文件夹 and Extensions folder 7-44
系统示警声 system alert sounds 6-8, 6-22
系统事件屏蔽参数 system event masks 2-28至2-29, 2-99
系统文件 System file 7-44
系统文件的组织 organization of 7-41至7-45
系统文件夹 System Folder
的替身类型 alias type for 7-40
定义 defined 7-5
查找 finding 7-54
系统颜色表，系统颜色表和调色板管理器 system color tables, and Palette Manager 4-21
下拉式菜单 pull-down menus
创建 creating 3-43至3-52
定义的 defined 3-6
显示“剪贴板”(编辑菜单) Show Clipboard command (Edit menu) 3-25
显示矩形，示警框和对话框中的磁盘插入事件 display rectangles, for items in alert and dialog boxes 6-29, 6-32至6-35
“显示旁述”命令（帮助菜单） Show Balloons command (Help menu) 6-68
项目编号 item numbers 6-28
项目列表资源 item list resources
计算…中的项目数 counting items in 6-128
定义 defined 6-13
的实例 example of 6-27至6-28
的Rez 输入格式 Rez input format for 6-26至6-30
的Rez 输出格式 Rez output format for 6-151至6-156
指定对话框的 specifying for a dialog box 6-24
指定示警框的 specifying for an alert box 6-20
项目颜色表资源 item color table resources 6-158至6-164
消息字符串资源 message string resources 7-27至7-30
小 4- 位彩色图标资源 small 4-bit color icon resources
创建 creating 7-11, 7-24
的Rez 输出格式 Rez output format for 7-60至7-61
小 8 位彩色图标资源 small 8-bit color icon resources
创建 creating 7-11, 7-24
的Rez 输出格式 Rez output format for 7-62至7-63
小（16X16 像素）图标 small (16-by-16 pixel) icons
的资源标识符 resource IDs for 7-15
的资源 resources for 7-10
小图标列表资源 small icon list resources
创建 creating 7-11, 7-15, 7-24
的Rez 输出格式 Rez output format for 7-58至7-59
校准点 hot spot 2-62
协同多任务 cooperative multitasking 2-10
“新文稿”命令（文件菜单） New command (File menu) 3-23

信息窗口　information windows
在…中使用固定样板底板核查框　using the stationery checkbox in 7-34
使用…自定义图标　using至customize icon 7-17
中的版本信息　version information in 7-8，7-32
信息方框　Info box。参见信息窗口
修改键　modifier keys 2-20至2-21，2-81，3-77
询问文稿　query documents
为…创建图标　creating icons for 7-16，7-17
的默认图标　default icon for 7-12
定义　defined 7-4

Y

颜色窗口记录　color window records 4-65至4-68
颜色控制板　Color control panel 4-20，4-65
掩码　masks
事件屏蔽参数　event。参见事件屏蔽参数
掩码图标　icon 7-13，7-15
样式　styles
改变菜单项目的样式　changing a menu item's 3-47，3-60
菜单项目的样式　of a menu item 3-12
移动鼠标事件　mouse-moved events 2-23，2-62至2-67，2-86
"页面设置"命令（文件菜单）　Page Setup command (File menu)　3-23
"隐藏夹纸板"命令（编辑菜单）　Hide Clipboard command　(Edit menu)　3-25
"隐藏旁述"命令（帮助菜单）　Hide Balloons command (Help menu)　6-68
"隐藏其它"命令（应用程序菜单）　Hide Others command　(Application menu)　3-33
应用程序　applications
的替身类型　alias type for 7-40
在苹果菜单项目文件夹中　in Apple Menu Items folder 7-43
为...创建图标　creating icons for 7-11至7-17
默认图标　default icon for 7-12
从 Finder 启动　launching from the Finder 7-25至7-26
共享　sharing 2-68
签名的署名　signatures for 7-8至7-10
之间的切换　switching between 2-5
应用程序版本信息　version information for applications
Finder 窗口中的　in Finder windows 7-31
信息窗口中的　in information windows 7-32
署名资源中的　in signature resource 7-8
'vers'资源中的　in 'vers' resource 7-31至7-32
应用程序不存在示警框售信息　application-unavailable alert box messages 7-27至7-30
应用程序菜单　Application menu 3-33
从示警框和对话框存取　accessing from alert and dialog boxes 6-68至6-69
"隐藏其它"命令　Hide Others command 3-33
应用程序的名称字符串资源　name string resources for applications 7-27至7-30
应用程序定义的项目　application-defined items
创建　creating 6-56至6-61
在示警框或对话框中说明　specifying in alert or dialog boxes 6-29至6-30，6-155
应用程序堆　application heap 2-32
应用程序图标　application icons。参见图标组
创建　creating 7-10至7-17
默认　default 7-12
应用程序未找到信息字符串资源　application-missing message string resources 7-27至7-30，7-68至7-69
硬盘，及其替身类型　hard disks，alias type for 7-40
用户界面准则　user interface guidelines
控制管理器的　for Control Manager 5-52至5-53
控制部件的　for controls 5-5至5-13
对话管理器的　for Dialog Manager 6-8至6-12，6-31至6-34，6-37至6-41，6-42，6-62至6-64
图标的　for icons 7-13，7-15
静态文本项目的　for static text items 6-40至6-41
窗口的　for windows 4-5至4-7，4-21，4-30至4-32
语系系统资源集合及其图标　script system resource collections，icon for 7-37
语音资源　sound resources
的图标　icon for 7-37至7-38
安装和删除　installing and removing 7-41
预置选择文件　preferences files 7-43
预置选择文件夹　Preferences folder
的替身类型　alias type for 7-40

定义 defined 7-43
查找 finding 7-54
在…中储存文件 storing files in 7-43
约束位 bundle bit 7-34
约束资源 bundle resources 7-20至7-25，7-45，7-65至7-68
阅显示矩形 refCon 域 refCon field 4-111

Z

主切换 major switches 2-16
"注意"示警框 note alerts
桌面 desktop 7-3
桌面对象 desktop objects 3-20
桌面附件 desk accessories
处理…中的事件 handling events in 2-36，2-87，2-95
和系统 and System 7 7-13
桌面附件资源 desk accessory resource。参见'DRVR'资源类型
桌面管理器 Desktop Manager 7-45
桌面数据库 desktop database 7-5，7-45至7-46
桌面图案 desktop pattern 4-112
桌面文件 Desktop file 7-5
桌面文件夹 Desktop Folder
定义 defined 7-43
查找 finding 7-54
资源 resources 1-11至1-13
警告资源 alert 6-19至6-22，6-150至6-151
警告颜色表资源 alert color table 6-157至6-158
替身记录 alias record 7-40
应用程序未找到消息字符串资源 application-missing message string 7-68至7-69
束资源 bundle 7-20至7-25，7-65至7-68
彩色图标资源 color icon 3-62，7-64
控制部件资源 control 5-18至5-28，5-118至5-121
控制部件颜色表 control color table 5-121至5-123
控制部件定义函数 control definition function 5-14，5-109至5-115
定义资源 defined 1-11
对话资源 dialog 6-15，6-24至6-26，6-148至6-149
对话颜色表资源 dialog color table 6-75至6-76，6-156至6-157
驱动程序资源 driver 3-20，3-68，3-129，3-130
文件参考资源 file reference 7-18至7-25，7-64至7-65
文件夹资源 folder 7-55
字体资源 font 3-69
字库资源 font family 3-69
图标资源 icon 3-62，7-63
图标列表资源 icon list 7-13，7-57至7-58
国际配置资源 international configuration 3-32
项目颜色表资源 item color table 6-158至6-164
项目列表资源 item list。参见项目列表资源
键盘布局资源 keyboard-layout 2-39
键映象 key-map 2-39，2-41，2-42
键重映象 key-remap 2-39，2-41
大 8 位彩色图标资源 large 8-bit color icon 7-61至7-62
大 4- 位彩色图标资源 large 4-bit color icon 7-59至7-60
菜单资源 menu 3-151至3-154
菜单条资源 menu bar 3-155
菜单条定义函数 menu bar definition function 3-9，3-87，3-104
菜单颜色信息表 menu color information table 3-155至3-157
菜单定义过程 menu definition procedure 3-157
未找到应用程序名字字符串 missing-application name string 7-68
可移动资源 movable 7-36至7-38
署名资源 signature 7-8至7-10
大小资源 size 2-115至2-119，7-35
小 8 位彩色图标资源 small 8-bit color icon 7-62至7-63
小 4- 位彩色图标资源 small 4-bit color icon 7-60至7-61
小图标资源 small icon 3-62
小图标列表资源 small icon list 7-58至7-59
语音资源 sound 7-37至7-38
字符串资源 string 3-60
字符串列表资源 string list 3-60，3-65至3-67

系统文件资源,系统文件中的资源　System file, in the 1-12
Finder使用的资源　used by the Finder 7-6至7-32
版本资源　version 7-31至7-32，7-69至7-70
窗口资源　window 4-22，4-25至4-27，4-124至4-127
窗口颜色表资源　window color table 4-127至4-129
窗口定义函数资源　window definition function 4-22，4-120，4-127
资源类型　resource types 1-11
'actb'资源　'actb'　6-157至6-158
'alis'资源　'alis'　7-40
'ALRT'资源　'ALRT'。参见警告资源
'BNDL'资源　'BNDL'　7-20至7-25，7-65至7-68
'cctb'资源　'cctb'　5-121至5-123
'CDEF'资源　'CDEF'　5-14，5-109至5-115
'cicn'资源　'cicn'　3-62，3-154，7-64
'CNTL'资源　'cicn'　5-18至5-28，5-118至5-121
'dctb'资源　'dctb'　6-156至6-157
'DITL'资源　'DITL'。参见项目列表资源
'DLOG'资源　'DLOG'。参见对话资源
'DRVR'资源　'DRVR'3-20，3-68，3-129，3-130
'FKEY'资源　'FKEY'　2-39，3-118
'fld#'资源　'fld#'7-55
'FOND'资源　'FOND'3-69，3-129，3-130
'FONT'资源　'FONT'　3-69，3-129，3-130
'FREF'资源　'FREF'　7-18至7-25，7-64至7-65
'hmmu'资源　'hmmu'　3-44，3-68
'icl8'资源　'icl8'　7-61至7-62
'icl4'资源　'icl4'7-59至7-60
'ICN#'资源　'ICN#'　7-20，7-57至7-58
'ICON'资源　'ICON'　3-62，3-154，7-63
'ics8'资源　'ics8'　7-62至7-63
'ics4'资源　'ics4'　7-60至7-61
'ics#'资源　'ics#'7-15，7-58至7-59
'ictb'资源　'ictb'　6-158至6-164
'itlc'资源　'itlc'3-32
'itlk'资源　'itlk'　2-41
'KCHR'资源　'KCHR'　2-39至2-42
'KMAP'资源　'KMAP'　2-39至2-42
'MBAR'资源　'MBAR'　3-155
'MBDF'资源　'MBDF'　3-9，3-87，3-104
'mctb'资源　'mctb'　3-155至3-157
'MDEF'资源　'MDEF'　3-157
'MENU'资源　'MENU'　3-151至3-154
'SICN'资源　'SICN'　3-62，3-154
'SIZE'资源　'SIZE'2-115至2-119，7-35
'STR'资源　'STR'　3-60，7-27至7-30
'STR#'资源　'STR#'　3-60，3-65至3-67
'vers'资源　'vers'　7-31至7-32，7-69至7-70
'wctb'资源　'wctb'　4-127至4-129
'WDEF'资源　'WDEF'　4-22，4-120，4-127
'WIND'资源　'WIND'　4-22，4-25至4-27，4-124至4-127
资源标识符　resource IDs
　光标的　for cursors 2-63
　定义的　defined 1-11
　菜单项目的图符的　for icons in menu items 3-62
资源分支　resource forks 1-11
资源管理器　Resource Manager 1-11至1-13
　和控制管理器　and Control Manager 5-3
　和　and the Finder 7-3
资源映象　resource maps 1-12
子菜单　submenus
　创建　creating 3-53至3-56
　定义　defined 3-6
自定义示警框　custom alert boxes
　用Alret函数创建示警框　creating with the Alert function 6-106至6-108
　定义示警框　defined 6-9
自动键事件　auto-key events 2-38
自动键阈值速率　auto-key threshold rate 2-38
字符代码　character codes 2-40
字体　fonts
　点阵　bitmapped 3-27
　字体箱　suitcases for 7-37
　TrueType字体　TrueType 3-27
字体菜单　Font menu 3-26至3-27，3-69至3-70
字体文件夹　Fonts folder 7-44
字体资源　font resources
　的图标　icon for 7-37至7-38
　安装和删除　installing and removing 7-41
字形菜单　Style menu 3-15

的实例 example of 3-15
字形和字形菜单 and the Font menu 3-27
字的等效组合键 keyboard equivalents for 3-19
坐标系 coordinate systems 4-17至4-18

索引

（以英文字母为序）

符号

'****' file type ‘****’文件类型 7-20
'????' file type ‘????’文件类型 7-20

数字

4-bit color icons 4 位彩色图标 7-11
8-bit color icons 8 位彩色图标 7-11
16-by-16 pixel (small) icons 16X16 像素（小）图标 7-11，7-15
32-by-32 pixel (large) icons 32X32 像素（大）图标 7-11，7-13

A

About command (Apple menu) 关于命令（苹果菜单）3-21至3-22，3-45
acceptAppDiedEvents flag acceptAppDieEvents 标志 2-118
AcceptHighLevelEvent function AcceptHighLevelEvent 函数 2-69，2-70至2-71，2-87，2-90至2-92
acceptSuspendResumeEvents flag acceptSuspendResumeEvents 标志 2-117
ACount global variable ACount 全局变量 6-172
'actb' resource type ‘actb’资源类型 6-157至6-158
action procedures 动作过程
  for buttons，checkboxes，and radio buttons 关于按钮、复选框和单选按钮 5-36
  changing 更改 5-102
  creating 创建 5-115至5-117
  defined 定义 5-35
  for pop-up menus 关于弹出式菜单 5-36至5-37
  for scroll bars 关于滚动条 5-58至5-61
  specifying 说明 5-91，5-92
activate events 激活事件
  in alert boxes 在示警框中 6-82
  deactivating windows for alert and dialog boxes 失活示警框和对话框 6-64至6-66
  defined 定义 2-8
  in dialog boxes 在对话框中 6-85，6-97至6-100
  handling 处理 2-50至2-55，4-50至4-53
  and windows 和窗口 4-42
active controls 活动控制部件 5-11至5-13，6-13至6-14
active windows 活动窗口 4-6至4-7
AddResMenu procedure AddResMenu 过程。参见 AppendResMenu 过程
AEIMP (Apple Event Interprocess Messaging Protocol) AEIMP（苹果事件进程间通信协议） 2-13，2-67
AEProcessAppleEvent function AEProcessAppleEvent 函数 2-68，2-78
AESend function AESend 函数 2-73，2-78，2-100
A5 world A5 区域
  and context switching 上下文切换 2-16
  size of 大小 2-32
alert boxes 示警框
  accessing menus from 存取菜单 6-68至6-73
  creating 创建 6-18至6-23
  default colors of，changing 系统默认颜色，改变 6-157至6-158
  defined 定义 6-6
  dialog boxes，as distinguished from 与对话框的差异 6-6至6-8
  displaying 显示 6-22，6-61至6-75
  event filter function for 的事件过滤函数 6-86至6-89，6-145至6-147
  events in 中的事件 2-29至2-30，6-81至6-82
  items in 中的项目。参见示警框和对话框中的项目
  stages for 阶段 6-20至6-22，6-150至6-151
  types of 类型 6-8至6-9
  as windows 作为窗口 6-15至6-17
  window types for 窗口类型 4-9
alert color table resources 警告颜色表资源 6-157至6-158
Alert function Alert 函数 6-106至6-108
alert resources 警告资源
  defined 定义 6-15
  example of 的实例 6-20，6-21
  Rez input format for 的Rez 输入格式 6-19

至6-22
Rez output format for 的Rez 输出格式 6-150 至6-151
alerts 警告
creating 创建 6-18至6-23，6-105至6-112
defined 定义的 6-5
defining alert sounds for 定义示警声 6-22，6-144至6-145
displaying 显示 6-61至6-75
events in 中的事件 6-81至6-82
stages for 的阶段 -20至6-22，6-150至6-151
types of 的类型 6-8至6-9
using the system alert sound 使用系统示警声的 6-8
alert sounds 示警声
creating 创建 6-18至6-22
defined 定义的 6-6
defining 定义 6-22，6-144至6-145
specifying 说明 6-151
stages for 示警声的阶段 6-20至6-22，6-150至6-151
and the system alert sound 和系统示警声 6-8
aliases 替身
defined 定义 7-39
resolving 还原 7-39至7-41，7-51至7-53
alias files 替身文件 7-39
alias records 替身记录 7-40
alias types 替身类型 7-40
'alis' resource type ‘alis’资源类型 7-40
'ALRT' resource type ‘ALRT’资源类型。参见警告资源
altDBoxProc window type altDBoxProc 窗口类型 4-9
'amnu' folder type ‘amnu’文件夹类型 7-54
ANumber global variable ANumber 全局变量 6-172
'appe' file type ‘appe’文件类型 7-44
AppendDITL procedure AppendDITL 过程 6-54，6-125至6-127
AppendMenu procedure AppendMenu 过程 3-64至3-67，3-124至3-125
AppendResMenu procedure AppendResMenu 过程 3-20，3-68至3-70，3-128至3-129
Apple Event Interprocess Messaging Protocol (AEIMP) 苹果事件进程间通信协议 2-13，2-67
Apple events 苹果事件
application sending to itself 应用程序发给自己的 3-71
defined 定义 2-13
and the 和 inder 7-25至7-26
handling 处理 2-78
as high-level events 作为高级事件 2-67至2-69
Open Documents 打开文稿 7-20
sending 发送 2-78
in response至menu commands 响应菜单命令 3-71
Apple menu 苹果菜单 3-20
About command “关于”命令 3-21至3-22，3-45
accessing from dialog boxes 从对话框存取 6-68至6-73
adding items to 添加项目到 3-68至3-69
creating 创建 3-43至3-44
handling 处理 3-80至3-81
Apple Menu Items folder 苹果菜单项目文件夹
adding items from, to Apple menu 从…向苹果菜单添加项目 3-20至3-21
alias type for 的替身类型 7-40
defined 定义的 7-43
finding 查找 7-54
'APPL' file type ‘APPL’文件类型 7-10
application-defined items 应用程序定义的项目
creating 创建 6-56至6-61
specifying in alert or dialog boxes 在示警框或对话框中说明 6-29至6-30，6-155
application heap 应用程序堆 2-32
application icons 应用程序图标。参见图标组
creating 创建 7-10至7-17
default 默认 7-12
Application menu 应用程序菜单 3-33
accessing from alert and dialog boxes 从示警框和对话框存取 6-68至6-69
Hide Others command “隐藏其它”命令 3-33
application-missing message string resources 应用程序未找到信息字符串资源 7-27至7-30，7-68至7-69
applications 应用程序
alias type for 的替身类型 7-40
in Apple Menu Items folder 在苹果菜单项目文件夹中 7-43
creating icons for 为...创建图标 7-11至7-17
default icon for 默认图标 7-12

launching from the Finder　从 Finder 启动　7-25 至 7-26
sharing　共享 2-68
signatures for　签名的署名 7-8至7-10
switching between　之间的切换 2-5
application-unavailable alert box messages　应用程序不存在示警框售信息 7-27 至 7-30
arrow cursor　箭头光标 2-63
arrows, in scroll bars　滚动条中的箭头。参见滚动箭头
'****' file type　'****' 文件类型 7-20
AtMenuBottom global variable　AtMenuBottom 全局变量 3-151
auto-key events　自动键事件 2-38
auto-key threshold rate　自动键阈值速率 2-38
A/UX and cancel events　A/UX 和取消事件 2-46
AuxCtlRec data type　AuxCtlRec 数据类型 5-76 至 5-77, 5-107
auxiliary control records　辅助控制记录 5-76 至 5-77, 5-107
auxiliary window record　辅助窗口记录 4-21, 4-73 至 4-74
AuxWinRec data type　AuxWinRec 数据类型 4-73 至 4-74

B

background applications, and Extensions folder　后台应用程序，及功能扩展文件夹 7-44
background processes　后台进程 2-4
Balloon Help online assistance, for icons 7-38　旁述联机帮助，旁述图标　7-38。参见旁述
BeginUpdate procedure　BeginUpdate 过程 2-47, 4-50, 4-106 至 4-107
'BNDL' resource type　'BNDL' 资源类型 7-20 至 7-25, 7-65 至 7-68
BringToFront procedure　BringToFront 过程 4-90
bundle bit　约束位 7-34
bundle resources　约束资源 7-20至7-25, 7-45, 7-65 至 7-68
Button function　Button 函数 2-108
buttons　按钮
active　活动按钮 5-11至5-13
control definition ID for　控制部件定义标识符 5-14
creating, in windows　创建，在窗口中 5-17 至 5-20
default　默认 6-10至6-11, 6-22, 6-30至6-31, 6-56 至 6-61
defined　定义的 5-5
events in　中的事件 5-31至5-37, 6-78
inactive　不活动　5-11至5-13
part code for　的部位代码 5-32
specifying in alert or dialog boxes　在示警框或对话框中说明　6-29 至 6-30, 6-153
titles for　的标题 6-37至6-39

C

CalcMenuSize procedure　CalcMenuSize 过程 3-143, 5-28
CalcVisBehind procedure　CalcVisBehind 过程 4-119
CalcVis procedure　CalcVis 过程 4-119
canBackground flag　canBackground 标志 -117
cancel events　取消事件 2-46至2-47
Can't Undo command (Edit menu)　"不能恢复"命令（编辑菜单）-24, 3-25
Caps Lock key　Caps Lock 键 2-20
caret, time between blinks of 2-113　插入符，闪烁间隔 3-24, 3-25
catalog files　目录文件
defined　定义　7-5
Finder information in　中的Finder信息 7-32 至 7-34, 7-46 至 7-51
CautionAlert function　CautionAlert 函数 6-111
caution alerts　"当心"示警框
creating with the CautionAlert function　用 CautionAlert 函数创建　6-111
defined　定义的　6-9
'cctb' resource type　'cctb' 资源类型 5-121至5-123
'CDEF' resource type　'CDEF' 资源类型 5-14, 5-109 至 5-115
character codes　字符代码 2-40
checkboxes　复选框
active　激活的　5-11至5-13
changing setting and display of　改变设定和显示 5-38 至 5-39
control definition ID for　的控制部件定义标识符 5-14
defined　定义的　5-5至5-6
events in　中的事件 6-78
in windows　窗口中的 5-31 至 5-32, 5-38 至 5-39

inactive 不活动的 5-11至5-13
part code for 复选框的部位代码 5-32
specifying in alert or dialog boxes 在示警框或对话框中说明 6-29至6-30，6-153
CheckItem procedure CheckItem 过程 3-61至3-62，3-136
CheckUpdate function CheckUpdate 函数 4-116
'cicn' resource type ‘cicn’资源类型 7-64
specifying a menu item's 指定菜单项目的 3-62，3-154
Clear command (Edit menu) “清除”命令（编辑菜单）3-25，6-69
ClearMenuBar procedure ClearMenuBar 过程 3-110
ClipAbove procedure ClipAbove 过程 4-116至4-117
Clipboard 剪贴板 3-24
converting data至and from 转换数据自和到 2-58至2-60
Hide Clipboard command (Edit menu) “隐藏剪贴板”命令（编辑菜单）3-25
Show Clipboard command (Edit menu) “显示剪贴板”命令（编辑命令）3-25
close box 关闭框 4-6
Close command (File menu) “关闭”命令（文件菜单）3-23
CloseDialog procedure CloseDialog 过程 6-119至6-120
close region 关闭区域 4-12
CloseWindow procedure GloseWindow 过程 4-60，4-104至4-105
closing windows 关闭窗口 4-60至4-62
routines for 的例程 4-103至4-106
'CNTL' resource type ‘CNTL’ 资源类型
example of 的实例 5-18，5-21，5-23，5-26
Rez input format for 的Rez 输入格式 5-18至5-24，5-25至5-28
Rez output format for 的Rez 输出格式 5-118至5-121
Color control panel 颜色控制板 4-20，4-65
color flag color 标志 7-48
color graphics ports 彩色图形端口 4-20
color icon resources 彩色图标资源 3-46，3-62，7-64
color icons 彩色图标 7-11，7-13至7-16
Color 彩色 QuickDraw
checking for 检查
when creating a window 创建窗口时 4-77
when zooming windows 缩放窗口时 4-54
and color window records 和彩色窗口记录 4-65
coordinate systems in 中的坐标系统 4-17至4-18
creating color windows 创建彩色窗口 4-20至4-21
and the Window Manager port 和窗口管理器端口 4-74
colorReserved flag ColorReserved 标志 7-48
colors 色彩
in alert boxes and dialog boxes 示警框和对话框中的 6-156至6-164
in controls 控制部件的 5-5
Label menu commands for 标签菜单命令 7-16
in menus 菜单中的 3-46，3-62，3-98至3-102
in windows 窗口中的 4-20至4-21
ColorSpec data type ColorSpec 数据类型 4-72，5-78至5-79
color window records 颜色窗口记录 4-65至4-68
Command key Command 键 2-20
Command-key equivalents 等效命令键。参见等效组合键
Command-Shift-number key sequences Command-Shift-数字键序列 2-39，2-86
content region 内容区域
defined 定义 4-6，4-12
drawing 绘制 4-39至4-40
updating 更新 4-40至4-41，4-48至4-50
context of a process, switching 进程内容，进程内容的切换 2-15至2-16
control color table records 控制色彩表记录 5-77至5-80
control color table resources 控制色彩表资源 5-121至5-123
control color tables 控制色彩表 6-160至6-162
control definition functions 控制部件定义函数
custom 自定义控制部件 5-109至5-115
defined 定义 5-14
for pop-up menus 弹出式菜单的 5-14，5-36
standard 标准 5-14
control definition IDs 控制部件定义标识符
for buttons 按钮的 5-14
for checkboxes 复选框的 5-14
for custom controls 自定义 5-109至5-110

defined 定义 5-14
for pop-up menus 弹出式菜单的 5-14
for radio buttons 单选按钮的 5-14
for scroll bars 滚动条的 5-14
specifying 设定 5-18，5-19，5-25，5-83 至 5-84，5-120
Control key Control 键 2-20
control list 控制部件列表 4-14
Control Manager 控制管理器 5-3 至 5-135
application-defined routines for 应用程序为定义的例程 5-109 至 5-117
data structures in 中的数据结构 5-72 至 5-80
and Dialog Manager 和对话管理器 5-3，5-5，5-16，5-29，5-31，6-78
and Event Manager 和事件管理器 5-3
and List Manager 和列表管理器 5-3
localization guidelines 本地化准则 5-15，5-19 至 5-20，5-85，5-120
and Resource Manager 和资源管理器 5-3
resources for 5-117 至 5-123 的资源 5-117 至 5-123。参见控制颜色表；控制定义函数；控制资源；
routines in 的例程 5-80 至 5-117
user interface guidelines 用户界面准则 5-5 至 5-13，5-52 至 5-53
and Window Manager 和窗口管理器 5-3
control panels，installing and removing 控制板，控制板的安装和删除 7-41
Control Panels folder 控制板文件夹
alias type for 的替身类型 7-40
defined 定义 7-43
finding 查找 7-54
ControlRecord data type ControlRecord 数据类型 5-73 至 5-75
control records 控制记录 5-73 至 5-75
control resources 控制资源
example of 的实例 5-18，5-21，5-23，5-26
Rez input format for 的Rez 输入格式 5-18 至 5-24，5-25 至 5-28
Rez output format for 的Rez 输出格式 5-118 至 5-121
controls 控制部件
action procedures for 的动作过程 5-91，5-102，5-115 至 5-117
activating 激活 5-13
active 活动 5-11 至 5-13，6-13
in alert and dialog boxes 示警框和对话框中的 6-29 至 6-30，6-154
colors in 的色彩 5-5，5-101，5-121 至 5-123
control definition IDs for 的控制定义标识符 5-18，5-19，5-25，5-83 至 5-84，5-120
custom 的定制。参见定制的控制功能
defined 定义 5-3
dragging 拖拽 5-35，5-99 至 5-100，5-114 至 5-115
drawing 绘制 5-17，5-85 至 5-88
events in 中的事件 5-88 至 5-93，6-78
highlight states of 的醒目显示状态 5-35，5-98 至 5-99
inactive 无效 5-13，6-13，6-74
indicators in 中的指示符。参见指示符
invisible 在不可见…中 5-18，5-19，5-25，5-83，5-97，5-119
mouse events in 中的鼠标事件 5-11 至 5-13，5-30 至 5-70
moving 移动 5-65 至 5-70，5-97 至 5-98
part codes for 的部位代码 5-31 至 5-32
pop-up menus 弹出式菜单的 6-42 至 6-44
rectangles for 5-18 至 5-19，5-22 至 5-23，5-25 至 5-26，5-82，5-118 的矩形 5-18 至 5-19，5-22 至 5-23，5-25 至 5-26，5-82，5-118。参见显示矩形
reference values for 的参考值
changing 改变…的参考值 5-105 至 5-106
determining 确定…的参考值 5-105
specifying 指定…的参考值 5-18，5-25，5-84，5-120
removing 删除 5-108 至 5-109
resizing 调整的…大小 5-65 至 5-70，5-98
resources for 的资源。参见控制功能资源
settings of 的设定状态
changing 改变…的设定状态 5-11，5-37 至 5-43，5-61，5-65 至 5-70，5-93 至 5-102
determining 确定…的设定状态 5-37 至 5-43，5-102 至 5-107
specifying initial 设定…的初始设定状态 5-18 至 5-19，5-21 至 5-23，5-25 至 5-28，5-83，5-118 至 5-119
standard 标准 5-4，5-15 至 5-28，5-81 至 5-85
text in 中的文字 5-18，5-19 至 5-20，5-25
titles of 的标题

changing 改变…的标题 5-96
determining 确定…的标题 5-104至5-105
specifying 设定…的标题 5-18，5-19至5-20，5-25，5-83，5-85，5-120，5-121
types of 5-4至5-13 的类型 5-4至5-13。参见按钮；核查框；自定控制功能；弹出式菜单；单选按钮；滚动条
updating 更新 5-29至5-30
visible 的可见性 5-18，5-19，5-25，5-83，5-119
in windows 窗口中的 4-14至4-15，4-23，x5-82
cooperative multitasking 协同多任务 2-10
coordinate systems 坐标系 4-17至4-18
Copy command (Edit menu) “拷贝”命令（编辑菜单）3-25，6-69
CountDITL function CountDITL 函数 6-128
CountMItems function CountMItem 函数 3-81，3-141
Create Publisher command (Edit menu) “创建版本”命令（编辑菜单）3-25
creators 创建者
assigning 指派 7-33至7-34，7-47至7-49
defined 定义 7-9至7-10
CtlCTab data type CtlCTab 数据类型 5-77至5-80
'ctrl' folder type ‘ctrl’文件夹类型 7-54
current menu list 当前菜单列表 3-40
current process 当前过程 2-28
cursors 光标
adjusting 调整…形状 2-65至2-67
arrow 箭头 2-63
cross 十字 2-63
I-beam I型 2-63
plus 加号 2-63
setting the appearance of 设定…形状 2-62
wristwatch 手表 2-63
custom alert boxes 自定义示警框
creating with the Alert function 用 Alret 函数创建示警框 6-106至6-108
defined 定义示警框 6-9
custom controls 定制控制部件 5-11，5-109至5-115
customized icons 定制图标 7-17至7-18
Cut command (Edit menu) “剪”命令（编辑菜单）3-25，6-69
CWindowPeek data type CWindowPeek 数据类型 4-65
CWindowPtr data type CWindowPtr 数据类型 4-65
CWindowRecord data type CWindowRecord 数据类型 4-65至4-68

D

DABeeper global variable DABeeper 全局变量 6-172
DAStrings global variable DAStrings 全局变量 6-172
data forks 数据分支 1-11
dBoxProc window type dBoxProc 窗口类型 2-16，4-9，6-25
'dctb' resource type ‘dctb’资源类型 6-156至6-157
default buttons 默认按钮 6-10至6-11，6-22，6-30至6-31，6-56至6-61
DeleteMCEntries procedure DeleteMCEntries 过程 3-147
DeleteMenuItem procedure DeleteMenuItem 过程 3-127至3-128
DeleteMenu procedure 3-109至3-110
DeleteMenu 过程。参见 DeleteMenuItem 过程
DelMCEntries procedure DelMCEntries 过程。参见 DeleteMCEntries 过程
DelMenuItem procedure DelMenuItem 过程。参见 DeleteMenuItem 过程
desk accessories 桌面附件
handling events in 处理…中的事件 2-36，2-87，2-95
and System 7 和系统 7-13
desk accessory resource 桌面附件资源。参见‘DRVR’资源类型
'desk' folder type ‘desk’文件夹类型 7-54
DeskHook global variable DeskHook全局变量 4-75
DeskPattern global variable DeskPattern 全局变量 4-113
desktop 桌面 7-3
desktop database 桌面数据库 7-5，7-45至7-46
Desktop file 桌面文件 7-5
Desktop Folder 桌面文件夹
defined 定义 7-43
finding 查找 7-54
Desktop Manager 桌面管理器 7-45
desktop objects 桌面对象 3-20
desktop pattern 桌面图案 4-112
dial controls 拨盘控制部件 5-11
dialog boxes 对话框
accessing menus from 存取…的菜单

3-84，6-68至6-73
alert boxes，as distinguished from 示警框和…的区别 6-6至6-8
closing 关闭 6-100至6-101
creating 创建 6-23至6-26，6-113至6-119
default colors of，changing 改变…的默认颜色 6-156至6-157
defined 定义 6-5
displaying 显示 6-61至6-75
disposing of 处理 6-113，6-119至6-120
event filter function for 的事件过滤函数 6-86至6-89
events in 中的事件 2-25，2-29至2-30，6-82至6-100，6-135至6-143
examples of 的实例 1-9
items in 中的项目。参见示警框和对话框中的项目
types of 的类型 6-9至6-12
as windows 作为窗口 6-15至6-17
window types for 的窗口类型 4-9至4-10
dialog color table resources 对话颜色表资源 6-75，6-156至6-157
DialogCopy procedure DialogCopy 过程 6-133
DialogCut procedure DialogCut 过程 6-132至6-133
DialogDelete procedure DialogDelete 过程 6-134
Dialog Manager 对话管理器 6-5至6-172
application-defined routines for 应用程序为…定义的例程 6-143至6-147
and Control Manager 控制管理器 5-3，5-5，5-16，5-29，5-31，6-78
data structures in 中的数据结构 6-101至6-102
initializing 初始化6-102至6-105
localization guidelines for 本地化准则 6-34至6-35，6-39，6-40，6-80，6-130
and Menu Manager 和菜单管理器 6-68至6-73
resources in 6-147至6-164 中的资源 6-147至6-164。亦参见警告颜色表资源；警告资源；对话颜色表资源；对话资源；项目颜色表资源；项目列表资源
routines in 的例程 6-102至6-147
testing for availability of AppendDITL，ShortenDITL，and ClountDITL routines 测试AppendDITL、ShortenDITL和CountDITL 的可用性 6-18
and 和 extEdit 6-79至6-80
user interface guidelines for 用户界面准则 6-8至6-12，6-31至6-34，6-37至6-41，6-42，6-62至6-64
and Window Manager 和窗口管理器 4-13至4-14，6-15至6-16
DialogPaste procedure DialogPaste 过程 6-133至6-134
DialogPeek data type DialogPeek 数据类型 6-102
DialogPtr data type DialogPtr 数据类型 6-102
DialogRecord data type DialogRecord 数据类型 6-101至6-102
dialog records 对话记录 6-101至6-102
dialog resources 对话资源
defined 定义 6-15
example of 的实例 6-24
Rez input format for 的Rez 输入格式 6-24至6-26
Rez output format for 的Rez 输出格式 6-148至6-149
DialogSelect function DialogSelect 函数 2-29，6-80，6-89至6-93，6-139至6-141
DIBadMount function DIBadMount 函数 2-56
DInfo data type DInfo 数据类型 7-50
directories，finding system-related 查找和系统有关的目录 7-53至7-56
directory information record 目录信息记录 7-50
disabled constant disabled 常数 3-8
disabled items 无效项目 6-13
DisableItem procedure 3-58至3-59，3-131至
DisableItem 过程 3-58至3-59，3-131， -132
'disk' file type 'disk'文件类型 7-20
disk-inserted events 磁盘插入事件
in alert and dialog boxes 示警框和对话框中的 6-87
handling 处理 2-55至2-56
and the ModalDialog procedure 和 ModalDialog过程 2-56
and standard file dialog boxes 和标准文件对话框 2-55
disks，opening 磁盘，打开磁盘插入事件 7-20
display rectangles，for items in alert and dialog boxes 显示矩形，示警框和对话框中的磁盘插入事件 6-29，6-32至6-35
DispMCInfo procedure DispMCInfo 过程。参见DisposeMCInfo 过程

DisposDialog procedure　DisposDialog 过程。参见 DisposeDialog 过程
DisposeControl procedure　DisposeControl 过程 5-108
DisposeDialog procedure　DisposeDialog 过程 6-120
DisposeMCInfo procedure　DisposeMCInfo 过程 3-145
DisposeMenu procedure　DisposeMenu 过程 3-140
DisposeWindow procedure　DisposeWindow 过程 4-105 至 4-106
DITLMethod data type　DITLMethod 数据类型 6-51，6-166
'DITL' resource type　'DITL' 资源类型。参见项目列表资源
dividers　(菜单项目)分隔线 3-12
DlgCopy procedure　DlgCopy 过程。参见 DialogCopy 过程
DlgCut procedure　DlgCut 过程。参见 DialogCut 过程
DlgDelete procedure　DlgDelete 过程。参见 DialogDelete 过程
DlgFont global variable　DlgFont 全局变量 6-172
DlgPaste procedure　DlgPaste 过程。参见 DialogPaste 过程
'DLOG' resource type　'DLOG' 资源类型。参见对话资源
documentProc window type　documentProc 窗口类型 4-10
documents　文稿 5-9 至 5-10，5-43 至 5-70
  creating icons for　创建…的图标 7-16至7-17，7-17 至 7-18
  creators of　的创建者 7-9至7-10
  default icon for　的默认图标 7-12
  file types of　的文件类型 7-9 至 7-10，7-18 至 7-20
  opening from the Finder　从 Finder 打开 7-25 至 7-26
  prohibiting users from opening from Finder　禁止用户从 Finder 打开 7-30
  window types for　的 1 窗口类型 4-8
document windows　文稿窗口
  defined　定义 4-4
  positioning　的定位 4-30
  saving position of　存储…的位置 4-34
  window types for　的窗口类型 4-8
doesActivateOnFGSwitch flag　doesActivateOnFGSwitch 标志 2-117
double clicks, time between　双击，双击的时间间隔 2-38，2-113
DragControl procedure　DragControl 过程 5-99 至 5-100
DragGrayRgn function　DragGrayRgn 函数 4-96 至 4-98
DragHook global variable　DragHook 全局变量 4-98，4-100，4-102，4-104
drag region　拖拽区域 4-12
DragWindow procedure　DragWindow 过程 4-44，4-94 至 4-95
DrawControls procedure　DrawControls 过程 5-87 至 5-88
DrawDialog procedure　DrawDialog 过程 6-142
DrawGrowIcon procedure　DrawGrowIcon 过程 4-12，4-39，4-86 至 4-87
DrawMenuBar procedure　DrawMenuBar 过程 3-51，3-58，3-113
DrawNew procedure　DrawNew 过程 4-117
Draw1Control procedure　Draw1Control 过程 5-88
'drop' alias type　'drop' 替身类型 7-40
drop folder, alias type for　drop 文件夹及其替身类型 7-40
'DRVR' resource type　'DRVR' 资源类型 3-20，3-68，3-129，3-130
DXInfo data type　DXInfo 数据类型 7-50 至 7-51

E

editable text items　可编辑文字项目
  events in　中的事件 6-79至6-80
  getting text from　从…中获取文字 6-48 至 6-51，6-130 至 6-131
  highlighting default text in　醒目显示…中的默认文字 6-131 至 6-132
  setting text in　在…中设定文字 6-131，6-153
  specifying in dialog boxes　指定对话框中的 6-29 至 6-30，6-153
Edition Manager　出版数据管理器 2-14
editions　出版数据
  creating icons for　为…创建图标 7-16
  defined　定义 7-4
Edit menu　编辑菜单 3-24 至 3-26
  accessing from alert and dialog boxes　从示警框和对话框中存取 6-68 至 6-73
  adjusting　调整 3-75 至 3-77
  Can't Undo command　的"不能取消"命令 3-25
  Clear command　的"清除"命令 3-25，

6-69
Copy command 的“拷贝”命令 3-25，6-69
Create Publisher command 的“创建版本”命令 3-25
Cut command 的“剪切”命令 3-25，6-69
Hide Clipboard command 的“隐藏夹纸板”命令 3-25
Paste command 的“贴”命令 3-25，6-69
Publisher Options command 的“版本选项”命令 3-25
Select All command 的“全选”命令 3-25
Show Clipboard command 的“显示夹纸板”命令 3-25
Subscriber Options command 的“订户选项”命令 3-25
Subscribe至command 的 SubscribeTO 命令 3-25
Undo command 的“取消”命令 3-25，6-69
8-bit color icons 8 位彩色图标 7-11
'empt' folder type ‘empt’文件夹类型 7-54
enabled constant enabled 常数 3-8
enabled items 启用项目 6-13
EnableItem procedure EnableItem 过程 3-58 至3-59，3-131
EndUpdate procedure EndUpdate 过程 2-47，2-49，4-50，4-107
EraseRect procedure EraseRect 过程 4-103
ErrorSound procedure ErrorSound 过程 6-104
EventAvail function EventAvail 函数 2-21，2-88至2-89
event classes 事件类 2-68
event filter functions 事件过滤函数 6-86至6-89，6-145至6-147
event IDs 事件标识符 2-68
event loops 事件循环 2-24至2-25
Event Manager 事件管理器 2-3至2-132。参见操作系统事件管理器
application-definedroutinefor 应用程序为…定义的例程 2-114至2-115
and Control Manager 和控制管理器 5-3
data structures in 中的数据结构 2-79 至2-83
initializing 初始化2-17
and MultiFinder 和MultiFinder 2-7
and the PPC Toolbox 和 PPC 工具箱 2-7，2-13
and the Process Manager 和进程管理器 2-5
routines in 的例程 2-84至2-113
and the Scrap Manager 和剪贴区管理器 2-60
testing for features 功能测试 2-17
event masks 事件屏蔽参数
defined 定义 2-22，2-85
setting 设定 2-26至2-29
event messages 事件消息 2-19，2-80
event queue 事件队列。参见高级事件队列；操作系统事件队列
scanning for a cancel event in 扫描…中的取消事件 2-46
structure of 的结构 2-83
EventRecord data type EventRecord 数据类型 2-79 至2-81
event records 事件记录 2-79至2-81
defined 定义 2-4
and event loops 和事件循环 2-25
for high-level events 高级事件的 2-68，2-69
events 事件
activate 激活 2-8，2-50至2-55，4-50至4-53
in alert boxes 示警框中的 2-29至2-30，6-81至6-82，6-86至6-89，6-135
Apple 苹果。参见苹果事件
auto-key 自动键 2-8，2-38
in buttons in windows 窗口按钮中的 5-31至5-37
cancel 取消 2-46至2-47
in checkboxes in windows 窗口复选框中的 5-38至5-39
Command-period “命令”＋“.”组合键事件 2 -46
in controls 控制部件中的 5-11至5-13，5-30至5-70，5-88至5-93，6-78
defined 定义 2-4
in desk accessories 桌面附件中的 2-36
in dialog boxes 对话框中的 2-29至2-30，6-77至6-100，6-135至6-143
disk-inserted 磁盘插入 2-8，2-55至2-56
in editable text items 可编辑文字项目中的 6-79至6-80
filter function for，in alert and dialog boxes 示警框和对话框中的…过滤函数 6-86至6-89，

first click in an inactive window 非当前活动窗口中首次单击 4-42
getting information about 获得有关…的信息 2-18至2-21
in gray areas of scroll bars 滚动条灰度区域中的 5-57至5-61
handling 处理 2-26
high-level 高级。参见高级事件
interacting with the user from the background 从后台和用户交互作用的 2-12
keyboard 键盘 2-8，2-38至2-45
key-down 键按下 2-38
key-up 键释放 2-38，2-39
low-level 低级 2-8至2-9
masking out 屏蔽 2-27
in modal dialog boxes 模式对话框中的 6-82至6-89，6-135至6-137
in modeless dialog boxes 无模式对话框中的 2-25，6-89至6-100，6-138至6-141
and modifier keys 和修改键 2-20至2-21，2-34
mouse-down 2-8，2-33至2-3 按下鼠标 2-8，2-33至2-8
mouse-down in windows 窗口中的按下鼠标 4-42至4-45
mouse-moved 移动鼠标 2-12，2-23，2-62至2-67
mouse-up 释放鼠标 2-8，2-33
in movable modal dialog boxes 可移动模式对话框中的 6-89至6-100，6-138至6-141
null 空 2-9，2-57
operating-system 操作系统 2-10至2-12，2-58至2-67
in pop-up menus 弹出式菜单中的 6-42
in windows 窗口的弹出式菜单中的 5-31至5-37
priority of 优先级 2-15
processing 实例 2-21至2-30
in radio buttons in windows 窗口的单选按钮中的 5-31至5-32
received in the background 在后台收到的 2-12
receiving from other applications 从其它应用程序收到的 2-69至2-71
resume 继续执行 2-12，2-60至2-62
in scroll arrows 滚动箭头中的 5-57至5-61
in scroll bars 滚卷条中的 5-43至5-65
in scroll boxes 滚动框中的 5-53至5-57
sources of 源 2-6
suspend 暂停 2-12，2-60至2-62
switching contexts 交换内容的 2-15至2-16
types of 类型 2-5，2-18，2-79
update 更新 2-9，2-47至2-50，4-48至4-50
in windows 窗口中的 4-21至4-22，4-41至4-53
event types 类型 2-5，2-18，2-79
EvQEl data type EvQEl 数据类型 2-83
exported AppleShare folder, alias type for 输出的AppleShare 文件夹，…的替身类型 7-40
extended directory information record 扩展的目录信息记录 7-50至7-51
extended file information record 扩展的文件信息记录 7-49
Extensions folder 功能扩展文件夹
alias type for 的替身类型 7-40
defined 定义的 7-44
finding 查找 7-54
'extn' folder type 'extn'文件夹类型 7-54

F

'faam' alias type 'faam'替身类型 7-40
'fact' alias type 'fact'替身类型 7-40
'fadr' alias type 'fadr'替身类型 7-40
'faet' alias type 'faet'替身类型 7-40
'faex' alias type 'faex'替身类型 7-40
'famn' alias type 'famn'替身类型 7-40
'fapf' alias type 'fapf'替身类型 7-40
'fapn' alias type 'fapn'替身类型 7-40
'fash' alias type 'fash'替身类型 7-40
'fast' alias type 'fast'替身类型 7-40
'fasy' alias type 'fasy'替身类型 7-40
'fdrp' alias type 'fdrp'替身类型 7-40
'ffil' file type 'ffil'文件类型 7-37
file information record 文件信息记录 7-34，7-47至7-49
File menu 文件菜单 3-22至3-24
Close command 的“关闭”命令 3-23，7-18
New command 的“新文稿”命令 3-23
Open command 的“打开”命令 3-23
Page Setup command 的“页面设置”命令 3-23
Print command 的“打印”命令 3-23
Quit command 的“退出”命令 3-23，7-18

Save As command 的“存储为”命令 3-23
Save command 的“存储”命令 3-23
file reference resources 文件参考资源 7-18至 7-25，7-64至7-65
files 文件。参见应用程序、文稿
data fork of 数据分支 1-11
managing 管理 -15
resource fork of 资源分支 1-11
temporary 临时 7-43
user comments associated with 和…有关的用户评论 7-45
file types 文件类型
for alias files 替身文件的 7-40
'appe' ‘appe’文件类型 7-33，7-47
'APPL' ‘APPL’文件类型 7-10
assigning 的指派 7-9至7-10，7-33至 7-34，7-47至7-49
'****' ‘****’文件类型 7-20
defining in file reference resources 在文件参考资源中定义的 7-18至7-25
'DFIL' ‘DFIL’文件类型 7-37
'disk' ‘disk’文件类型 7-20
'FFIL' ‘FFIL’文件类型 7-37
'ffil' ‘ffil’文件类型-37
'fold' ‘fold’文件类型 -20
'ifil' ‘ifil’文件类型7-37
'kfil' ‘kfil’文件类型7-37
for movable resources 可移动资源的 7-37
opening documents of particular 打开特定…的文稿
from applications 从应用程序打开特定…的文稿 7-10
from the Finder 从 Finder 打开特定…的文稿 7-25至7-26
'????' ‘????’文件类型 -20
'scri' ‘scri’文件类型7-44
'sfil' ‘sfil’文件类型7-37
for stationery pads 固定样板文稿的 7-19
'tfil' ‘tfil’文件类型-37
filter function 过滤函数。参见事件过滤函数
for GetSpecificHighLevelEvent GetSpecificHighLevelEvent 的… 2-92至2-93，2-114至2-115
FindControl function FindControl 函数 5-33至5-35，5-44，5-53至5-54，5-58，5-89至5-90
FindDialogItem function FindDialogItem 函数 6-125
FindDItem function FindDItem 函数。参见 FindDialogItem函数
Finder
application interface to 的应用程序界面 7-3至7-78
database for a volume 卷的数据库 7-45至 7-46
data structures used by 使用的数据结构 7-46至7-51
defined 定义 7-3至7-5
and Process Manager 和进程管理器 7-25
relation至File Manager 和文件管理器的关系 7-3
and Resource Manager 和资源管理器 7-3
resources used by 使用的资源 7-6至7-32，7-56至7-70。参见应用程序未找到消息字符串资源；文件参考资源；图标列表资源；大8位彩色图标资源；大 4 位彩色图标资源未找到应用程序名字字符串资源；署名资源；小8位彩色图标资源；小 4位彩色图标资源；小图标列表资源；版本资源
routines for 的例程 7-51至7-56
user interface guidelines for 的用户界面准则 7-13，7-15
Finder flags Finder 标志 7-33至7-34，7-47至 7-48
Finder information, in the catalog file Finder 信息，目录文件中的—— 7-32至7-34，7-46至7-51
FindFolder function FindFolder 函数 7-54至7-56
FindWindow function FindWindow 函数 4-12，4-44，4-91至4-93，5-32至5-33
and activate events 和激活事件 2-51
and mouse events 和鼠标事件 2-33至2-37，4-42至4-44
FInfo data type FInfo数据类型 7-34，7-47至7-49
“first click” in an inactive window 非当前活动窗口中的“首次单击” 4-7，4-42
'FKEY' resource type ‘FKEY’资源类型 2-39，2-86，3-118
FlashMenuBar procedure FlashMenuBar 过程 3-141至3-142
'fld#' resource type ‘fld#’替身类型 7-55
floppy disks, alias type for 软盘，...的替身类型 7-40
'flpy' alias type ‘flpy’替身类型 7-40
FlushEvents procedure FlushEvents 过程 2-93至 2-94

folder resources 文件夹资源 7-55
folders 文件夹
  alias type for 的替身类型 7-40
  finding system-related 查找和系统有关的 7-53 至 7-56
folder types 文件夹类型 7-54
'fold' file type 'fold'文件类型 7-20
'FOND' resource type 'FOND' 资源类型 3-69
'font' folder type 'font'文件夹类型 7-54
Font menu 字体菜单 3-26 至 3-27，3-69 至 3-70
font resources 字体资源
  icon for 的图标 7-37 至 7-38
  installing and removing 安装和删除 7-41
'FONT' resource type 'FONT' 资源类型 3-69
fonts 字体
  bitmapped 点阵 3-27
  suitcases for 字体箱 7-37
  TrueType TrueType 字体 3-27
Fonts folder 字体文件夹 7-44
foreground process 前台进程 2-4
4-bit color icons 4 位彩色图标 7-11
frames，window 框架，窗口—— 4-6，4-12 至 4-13
'FREF' resource type 'FREF' 资源类型 7-18 至 7-25，7-64 至 7-65
FrontWindow function FrontWindow 函数 4-93 至 4-94
FSpGetFInfo function FSpGetFInfo 函数 7-34，7-49
FSpSetFInfo function FSpSetFInfo 函数 -34，7-49
FXInfo data type FXInfo 数据类型 7-49

G

GetAlertStage function GetAlertStage 函数 6-66，6-112
GetAlrtStage function GetAlrtStage 函数。参见 GetAlertStage 函数
GetAuxCtl function GetAuxCtl 函数。参见 GetAuxiliaryControlRecord 函数
GetAuxiliaryControlRecord function GetAuxiliaryControlRecord 函数 5-107
GetAuxWin function GetAuxWin 函数 4-115 至 4-116
GetCaretTime function GetCaretTime 函数 2-113
GetControlAction function GetControlAction 函数 5-106
GetControlMaximum function GetControlMaximum 函数 5-41 至 5-42，5-61，5-104
GetControlMinimum function GetControlMinimum 函数 5-103 至 5-104
GetControlReference function GetControlReference 函数 5-105
GetControlTitle procedure GetControlTitle 过程 5-104 至 5-105
GetControlValue function GetControlValue 函数 5-38 至 5-39，5-41 至 5-42，5-44，5-53 至 5-55，5-61，5-103
GetControlVariant function GetControlVariant 函数 5-106 至 5-107
GetCRefCon function GetCRefCon 函数。参见 GetControlReference 函数
GetCTitle procedure GetCTitle 过程。参见 GetControlTitle 过程
GetCtlAction function GetCtlAction 函数。参见 GetControlAction 函数
GetCtlMax function GetCtlMax 函数。参见 GetControlMaximum 函数
GetCtlMin function GetCtlMin 函数。参见 GetControlMinimum 函数
GetCtlValue function GetCtlValue 函数。参见 GetControlValue 函数
GetCVariant function GetCVariant 函数。参见 GetControlVariant 函数
GetCWMgrPort procedure GetCWMgrPort 过程 4-113 至 4-114
GetDblTime function GetDblTime 函数 2-113
GetDialogItem procedure GetDialogItem 过程 6-50，6-121 至 6-122
GetDialogItemText procedure GetDialogItemText 过程 6-50，6-130 至 6-131
GetDItem procedure GetDItem 过程。参见 GetDialogItem 过程
GetEvQHdr function GetEvQHdr 函数 2-100
getFrontClicks flag getFrontClicks 标志 2-118
GetGray function GetGray 函数 3-91
GetGrayRgn function GetGrayRgn 函数 4-113
GetItemCmd procedure GetItemCmd 过程 3-138 至 3-139
GetItemIcon procedure GetItemIcon 过程 3-137
GetItemMark procedure GetItemMark 过程 3-135
GetItem procedure GetItem 过程。参见 GetMenuItemText 过程
GetItemStyle procedure GetItemStyle 过程 3-133 至 3-134
GetIText procedure GetIText 过程。参见 GetDialogItemText 过程

GetKeys procedure GetKeys 过程 2-110
GetMBarHeight function GetMBarHeight 函数 3-113
GetMCEntry function GetMCEntry 函数 3-145 至 3-146
GetMCInfo function GetMCInfo 函数 3-52, 3-143 至 3-144
GetMenuBar function GetMenuBar 函数 3-112
GetMenu function GetMenu 函数 3-55, 3-106 至 3-108
GetMenuHandle function GetMenuHandle 函数 3-122
GetMenuItemText procedure GetMenuItemText 过程 3-80, 3-132
GetMHandle function GetMHandle 函数。参见 GetMenuHandle 函数
GetMouse procedure GetMouse 过程 2-108
GetNewControl function GetNewControl 函数 5-15 至 5-18, 5-24 至 5-25, 5-81 至 5-82
GetNewCWindow function GetNewCWindow 函数 4-28, 4-76 至 4-77
GetNewDialog function GetNewDialog 函数 6-113 至 6-115
GetNewMBar function GetNewMBar 函数 3-50 至 3-52, 3-111 至 3-112
GetNewWindow function GetNewWindow 函数 4-28, 4-78 至 4-79
GetNextEvent function GetNextEvent 函数 2-21, 2-89 至 2-90
GetOSEvent function GetOSEvent 函数 2-97 至 2-98
GetPortNameFromProcessSerialNumber function
GetPortNameFromProcessSeriealNumber 函数 2-107
GetProcessInformation function GetProcess-Information 函数 2-74
GetProcessSerialNumberFromPortName function
GetProcessSerialNumberFromPortName 函数 2-73, 2-82, 2-106
GetSpecificHighLevelEvent function
GetSpecificHighLevelEvent 函数 2-71, 2-77, 2-92 至 2-93
GetWindowPic function GetWindowPic 函数 4-110
GetWMgrPort procedure GetWMgrPort 过程 4-114
GetWRefCon function GetWRefCon 函数 4-111
GetWTitle procedure GetWTitle 过程 4-86
GetWVariant function GetWVariant 函数 4-24, 4-112
global coordinates 全局坐标 4-17
GlobalToLocal procedure GlobalToLocal 过程 5-34
graphics ports 图形端口 4-17至4-18
gray areas, in scroll bars 滚卷条中的灰度区域
  action procedures for 对于…的动作过程 5-58 至 5-61
  defined 定义 5-9
  events in 的事件 5-57至5-61
  part codes for 的部位代码 5-32
gray region 灰度区域 4-16
GrayRgn global variable GrayRgn 全局变量 4-16, 4-95, 4-113, 4-116
grow images, of windows 窗口中的图象增长 4-57, 4-87
GrowWindow function GrowWindow 函数 4-57 至 4-59, 4-99 至 4-100

H

hard disks, alias type for 硬盘，及其替身类型 7-40
hasBeenInited flag hasBeenInited 标志 7-48
hasBundle flag hasBundle 标志 7-17 至 7-18, 7-34
hasCustomIcon flag hasCustomIcon 标志 7-17 至 7-18, 7-34
hasNoINITS flag hasNoINITS 标志 7-48
'hdsk' alias type 'hdsk'替身类型 7-40
help balloon resources 旁述资源 3-44, 3-68
help balloons 旁述
  for icons 图标的 7-38
  for window frames 窗口框架的 4-23
help items, specifying in alert or dialog boxes 帮助项目，在示警框和对话框中指定的—— 6-29 至 6-30, 6-155 至 6-156
Help menu 帮助菜单 3-29 至 3-31
  accessing from alert and dialog boxes 从示警框和对话框存取 6-68 至 6-69
  adding items to 向…添加项目 3-67至3-68
  handling 处理 3-81
  Hide Balloons command 的“隐藏旁述”命令 6-68
  Show Balloons command 的“显示旁述”命令 6-68
Hide Balloons command (Help menu) “隐藏旁述”

命令（帮助菜单）6-68
Hide Clipboard command (Edit menu) “隐藏夹纸板”命令（编辑菜单）3-25
HideControl procedure HideControl 过程 5-39至5-40，5-65至5-67，5-97
HideDialogItem procedure HideDialogItem 过程 6-123至6-124
HideDItem procedure HideDItem 过程。参见 HideDialogItem 过程
Hide Others command (Application menu) “隐藏其它”命令（应用程序菜单）3-33
HideWindow procedure HideWindow 过程 4-61，4-89
hierarchical menus 嵌套式菜单 3-38，3-53至3-56
high-level event message record 高级事件消息记录 2-72，2-82至2-83
HighLevelEventMsg data type HighLevelEventMsg 数据类型 2-72，2-82
high-level event queue 高级事件队列 2-7，2-13
high-level events 高级事件 2-13至2-15，2-67至2-78。参见苹果事件
  accepting 的接受 2-69，2-70
  defined 定义 2-13
  determining the sender of 确定…的发送程序 2-72
  event classes 的事件类 2-68
  event IDs 的事件标识符 2-68
  handling 处理 2-67至2-78
  posting options 投寄选项 2-75
  receiving 接收 2-68，2-87
  replying to 应答 2-74
  and return receipts 和回执 2-77
  searching for a specific event 在…中搜索指定的事件 2-71至2-72，2-92
  sending 发送 2-73至2-77，2-101
  supporting 支持 2-25
  testing for availability 测试…的可用性 2-7
HiliteControl procedure HiliteControl 过程 5-98至5-99，6-13至6-14，6-50至6-51，6-74，6-98
HiliteMenu procedure HiliteMenu 过程 3-71，3-79，3-119至3-120
HiliteWindow procedure HiliteWindow过程 4-90
HiWord function HiWord 函数 4-58
HMGetHelpMenuHandle function HMGetHelpMenuHandle 函数 3-67至3-68，3-81，3-123至3-124
'hmmu' resource type ‘hmmu’资源类型 3-44，3-68
HMSetMenuResID procedure HMSetMenuResID 过程 6-71
hot spot 校准点 2-62
human interface guidelines 人机界面准则。参见用户界面准则

I，J

I-beam cursor I型光标 2-63
'icl8' resource type ‘icl8’资源类型
  creating 创建 7-11，7-24
  Rez output format for 的Rez 输出格式 7-61至7-62
'icl4' resource type ‘icl4’资源类型
  creating 创建 7-11，7-24
  Rez output format for 的Rez 输出格式 7-59至7-60
'ICN#' resource type ‘ICN#’资源类型
  creating 创建 7-11至7-15
  example of 的实例 7-14至7-15
  Rez output format for 的Rez 输出格式 7-57至7-58
  specifying in a bundle resource 在束资源中指定 7-20
icon families 图标库 7-11
icon list resources 图标列表资源
  creating 创建 7-11至7-15
  example of 图标 7-14至7-15
  Rez output format for 的Rez 输出格式 7-57至7-58
  specifying in bundle resources 在束资源中指定 7-20
icon masks 图标掩码 7-13，7-15
icon resources ('ICON' resource type) 图标资源（‘ICON’资源类型） 7-63
'ICON' resource type ‘ICON’资源类型 7-63
  specifying a menu item's 指定菜单项目的 3-46，3-62，3-154
icons 图标。亦参见图标库
  for applications 应用程序的 7-10至7-15
  black and white 黑白 7-13，7-15
  color 彩色 7-15至7-16
  creating 创建 7-11至7-18
  customizing 自定义 -17至7-18
  defaults used by Finder Finder使用的默认 -12，7-40
  defined 定义 7-3
  for documents 文稿的 7-16至7-18
  for editions 出版数据的 7-16

8-bit color 8 位彩色 7-11
for font resources 字体资源的 7-37
4-bit color 4 位彩色 7-11
for keyboard layout resources 键盘布局资源的 7-37
large 大图标 7-10，7-13
for menu items 菜单项目的 3-62至3-64
in menus 菜单中的 3-12，3-62至3-64，3-154
for query documents 询问文稿的 7-16
reduced 缩小的 3-62至3-63
for script system resource collections 语系系统资源集合的 7-37
small 小图标 7-13，7-15
for sound resources 语音资源的 7-37
specifying in alert or dialog boxes 示警框和对话框中指定的 6-29至6-30，6-154
for stationery pads 固定样板文稿的 7-16，7-19，7-36
system 系统 7-12
for TrueType font resources TrueType 字体资源 7-37
'ics8' resource type ‘ics8'资源类型
creating 创建 7-11，7-24
Rez output format for 的Rez 输出格式 7-62至7-63
'ics4' resource type ‘ics4'资源类型
creating 创建 7-11，7-24
Rez output format for 的Rez 输出格式 7-60至7-61
'ics#' resource type ‘ics#'资源类型
creating 创建 7-11，7-15，7-24
Rez output format for 的Rez 输出格式 7-58至7-59
'ictb' resource type ‘ictb'资源类型 6-158至6-164
'ifil' file type ‘ifil'文件类型 7-37
inactive controls 无效控制部件 5-13，6-13至6-14
inactive windows 非当前活动窗口 4-6至4-7
handling mouse-down events in 处理…中的鼠标按下事件 4-43
indicators, in controls 控制部件中指示符
defined 定义 5-11
dragging 拖拽 5-35，5-99至5-100，5-114至5-115
Info box 信息方框。参见信息窗口
information windows 信息窗口
using the stationery checkbox in 在…中使用固定样板底板核查框 7-34
using至customize icon 使用…自定义图标 7-17
version information in 中的版本信息 7-8，7-32
InitDialogs procedure InitDialogs 过程 6-103至6-104
'INIT' file type ‘INIT'文件类型 7-44
InitMenus procedure InitMenu 过程 3-103至3-104
InitProcMenu procedure InitProcMenu 过程 3-104
InitWindows procedure InitWindows 过程 4-75
input methods 输入方法 3-32
InsertMenuItem procedure InsertMenuItem 过程 3-64至3-67，3-126至3-127
InsertMenu procedure InsertMenu 过程 3-55，3-108至3-109
InsertResMenu procedure InsertResMenu 过程 3-69，3-129至3-130
InsMenuItem procedure InsMenuItem 过程。参见 InsertMenuItem 过程
InvalMenuBar procedure InvalMenuBar 过程 3-114
InvalRect procedure InvalRect 过程 4-107至4-108
InvalRgn procedure InvalRgn 过程 4-59，4-108
IPCListPorts function IPCListPorts 函数 2-74
isAlias flag isAlias 标志 7-40
IsDialogEvent function IsDialogEvent 函数 2-29，6-77，6-138至6-139
isHighLevelEventAware flag isHighLevelEventAware 标志 2-119
isInvisible flag isInvisible 标志 7-34
isOnDesk flag isOnDesk 标志 7-48
isShared flag isShared 标志 7-34，7-48
isStationeryAware flag isStationeryAware 标志 2-119
isStationery flag isStationery 标志 7-34
is32BitCompatible flag is32BitCompatible 标志 2-118
item color table resources 项目颜色表资源 6-158至6-164
item list resources 项目列表资源
counting items in 计算…中的项目数 6-128
defined 定义 6-13
example of 的实例 6-27至6-28
Rez input format for 的Rez 输入格式 6-26至6-30
Rez output format for 的Rez 输出格式

6-151 至 6-156
specifying for a dialog box 指定对话框的 6-24
specifying for an alert box 指定示警框的 6-20
item numbers 项目编号 6-28
items in alert and dialog boxes 示警框和对话框中的项目。参见定义应用程序的项目；按钮；核查框；控制功能；可编辑文字项目；项目列表资源，图片；弹出式菜单；单选按钮；静态文字项目
adding 添加 6-51 至 6-56，6-125 至 6-127
copying editable text from 从…拷贝可编辑文字 6-133
counting 对…计数 6-128
creating 创建 6-26 至 6-44
cutting editable text from 从…剪切可编辑文字 6-132 至 6-133
default colors of, changing 的默认颜色，改变…的默认颜色 6-158 至 6-164
defined 定义 6-13
deleting editable text from 从…中删除可编辑文字 6-134
display rectangles for 为…显示热血 6-29，6-32 至 6-35
drawing application-defined 绘制应用程序定义的 6-143 至 6-144
enabled and disabled 启用的和停用的 6-13，6-36
finding item numbers of 查找…的项目编号 6-125
getting handles to 获取…的间接指针 6-121 至 6-122
getting text strings from 从…中获取文字字符串 6-130 至 6-131
hiding 隐藏 6-123 至 6-124
highlighting default text in 醒目显示…中的默认文字 6-131 至 6-132
item numbers for 的项目编号 6-28
keyboard navigation 键盘引导的 6-44
manipulating 操纵 6-44 至 6-56，6-120 至 6-128
pasting editable text into 向…贴可编辑文字 6-133 至 6-134
pictures 的图片 6-29 至 6-30
redisplaying after hiding 隐藏…后重新显示 6-124
removing 删除 6-127 至 6-128
setting or changing 设定或改变 6-122 至 6-123
setting text strings in 在…中设定文字字符串 6-131
substituting text in 替换…中的文字 6-129 至 6-130
types of 的类型 6-13 至 6-14，6-30
'itlc' resource type 'itlc'资源类型 3-32，3-33
'itlk' resource type 'itlk'资源类型 2-41

K

'KCHR' resource type 'KCHR' 资源类型 2-39 至 2-42，3-17
keyboard equivalents 等效组合键 3-18 至 3-19
commonly used 通常使用的 3-19
defining 定义 3-47
reserved 保留的 3-18
keyboard events 键盘事件
in alert boxes 示警框中的 6-86 至 6-89
in dialog boxes 对话框中的 6-79 至 6-80，6-86 至 6-89，6-94 至 6-97
handling 处理 2-38 至 2-45
in windows 窗口中的 4-47
keyboard layout resources 键盘布局资源 2-39 至 2-42，3-17，3-32，3-33
Keyboard menu 键盘菜单 3-32 至 3-33
accessing from alert and dialog boxes 从示警框和对话框中存取 6-68 至 6-69
added by Menu Manager 菜单管理器添加的 3-19
keyboard resources 键盘资源
icon for 的图标 7-37 至 7-38
'KCHR' 'KCHR' 键盘资源 2-39 至 2-42
'KMAP' 'KMAP' 键盘资源 2-21，2-39 至 2-42
keyboards 键盘 2-40
Apple Extended Keyboard II, domestic 苹果扩展键盘 II，美国版的键盘 2-43
Apple Extended Keyboard II, ISO 苹果扩展键盘 II，ISO 版的键盘 2-43
Apple Keyboard II, ISO 苹果键盘 II，ISO 版的键盘 2-42
getting the state of 获得键盘状态 2-110
key-down events 键按下事件
defined 定义 2-8
handling keyboard equivalents 处理…的等效组合键 3-77 至 3-78
receiving 接收 2-38 至 2-46

KeyMap data type　KeyMap 数据类型 2-110
KeyTrans function　KeyTrans 函数。参见 KeyTranslate 函数
KeyTranslate function　KeyTranslate 函数 2-110至2-111，3-78
key-up events　键释放事件 2-8，2-29
'kfil' file type　'kfil'文件类型 7-37
KillControls procedure　KillControl 过程 5-108至5-109
'KMAP' resource type　'KMAP' 资源类型 2-21，2-39至2-42

L

Label menu commands　标签菜单命令 7-16
large 8-bit color icon resources　大8位彩色图标资源
　　creating　创建 7-11，7-24
　　Rez output format for　的Rez 输出格式 7-61至7-62
large 4-bit color icon resources　大4-位彩色图标资源
　　creating　创建 7-11，7-24
　　Rez output format for　的Rez 输出格式 7-59至7-60
large (32-by-32 pixel) icons　大（32X32 像素）图标 -10，7-11
launching applications, from the Finder　从Finder启动应用程序 7-25至7-26
listings　列表。参见示例例程
List Manager, and Control Manager　列表管理器和控制管理器 5-3
localAndRemoteHLEvents flag　LocalAndRemoteHLEvents 标志 2-119
local coordinates　局部坐标 4-18
local IDs　局部 ID
　　in bundle resources and file reference resources　束资源和文件参考资源中的 7-19，7-22，7-24
　　for mapping resources　映射资源的 7-18至7-25
localization guidelines　本地化准则
　　for Control Manager　控制管理器的 5-15，5-19至5-20，5-85，5-120
　　for Dialog Manager　对话管理器的 6-34至6-35，6-39，6-40，6-80，6-130
location names　位置名称 2-73，2-75
low-level events　低级事件
　　handling　处理 2-32至2-57
　　receiving　接收 2-21
LoWord function　LoWord 函数 4-58

M

Macintosh Toolbox　Macintosh工具箱
　　features provided by　提供的功能 1-4
　　introduction to　介绍 1-3至1-16
'macs' folder type　'macs'文件夹类型 7-54
major switches　主切换 2-16
marking character in a menu item 3-12　菜单列表中的标记字符
masks　掩码
　　event　事件屏蔽参数。参见事件屏蔽参数
　　icon　掩码图标 7-13，7-15
MBarEnable global variable　MBarEnable 全局变量 3-121
'MBAR' resource type　'MBAR' 资源类型 3-50，3-155
'MBDF' resource type　'MBDF' 资源类型 3-9，3-87，3-104
MCEntry data type　MCEntry 数据类型 3-99
MCTable data type　MCTable 数据类型 3-99
MCTableHandle data type　MCTableHandle 数据类型 3-99
'mctb' resource type　'mctb' 资源类型 3-52，3-99，3-143，3-155至3-157
'MDEF' resource type　'MDEF' 资源类型 3-87，3-157
memory, allocating　分配存储器 1-15
menu bars　菜单条
　　accessing from alert and dialog boxes　从示警框和对话框存取 6-68至6-73
　　creating　创建 3-49至3-50
　　defined　定义 3-6，3-10
　　handling mouse-down events in　处理…中的鼠标按下事件 3-72
　　installing　安装3-41
menu bar definition function　菜单条定义函数 3-9，3-87，3-104
MenuChoice function　MenuChoice 函数 3-118至3-119
menu color information table　菜单颜色信息表 3-52，3-98至3-102
　　menu bar entry　的菜单条入口 3-100

menu item entry 的菜单项目入口 3-101
menu title entry 的菜单标题入口 3-101
resources 资源 3-155至3-157
menu commands 菜单命令。参见菜单项目
About (Apple menu) “关于”(苹果菜单) 3-21
Can't Undo (Edit menu) “不能取消”(编辑菜单) 3-25
for changing colors in the Finder 在Finder中改变色彩的 7-16
Clear (Edit menu) “清除”(编辑菜单) 3-25, 6-68至6-69
Close (File menu) “关闭”(文件菜单) 3-23
Copy (Edit menu) “拷贝”(编辑菜单) 3-25, 6-68至6-69
Create Publisher (Edit menu) “创建版本”(编辑菜单) 3-25
Cut (Edit menu) “剪”(编辑菜单) 3-25, 6-68至6-69
Hide Balloons (Help menu) “旁述”(帮助菜单) 6-68
Hide Clipboard (Edit menu) “隐藏夹纸板”(编辑菜单) 3-25
Hide Others (Application menu) “隐藏其它”(应用程序菜单) 3-33
keyboard equivalents for 的等效组合键 3-16至3-19, 3-154
New (File menu) “新文稿”(文件菜单) 3-23
None (Color menu) “无”(彩色菜单) 7-16
Open (File menu) “打开”(文件菜单) 3-23
Other (Size menu) “其它”(大小菜单) 3-28
Page Setup (File menu) “页面设置”(文件菜单) 3-23
Paste (Edit menu) “贴”(编辑菜单) 3-25, 6-68至6-69, 7-17
Print (File menu) “打印”(文件菜单) 3-23
Publisher Options (Edit menu) “版本选项”(编辑菜单) 3-25
Quit (File menu) “退出”(文件菜单) 3-23
Save (File menu) “存储”(文件菜单) 3-23
Save As (File menu) “存储为”(文件菜单) 3-23
Select All (Edit menu) “全选”(编辑菜单) 3-25
Show Balloons (Help menu) “显示旁述”(帮助菜单) 3-30, 6-68
Show Clipboard (Edit menu) “显示剪贴板” 3-25
Subscriber Options (Edit menu) “引用选项”(编辑菜单) 3-25
Subscribe To (Edit menu) “引用”(编辑菜单) 3-25
Undo (Edit menu) “取消”(编辑菜单) 3-25, 6-68至6-69
menu definition procedure 菜单定义过程 3-148至3-151
and the A5 register 和 A5 寄存器 3-150
defined 定义 3-9
drawing color menus 绘制彩色菜单的过程 3-91
handling scrolling in menus 在菜单中处理滚动的过程 3-91, 3-93
standard 标准 3-148
writing 编写 3-87至3-95
menu handle 菜单句柄 3-95, 3-122
MenuHandle data type MenuHandle 数据类型 3-95
menu ID 菜单标识符 3-40, 3-152
MenuInfo data type MenuInfo 数据类型 3-96至3-97
menu items 菜单项目 3-12至3-16
in an accumulating group 累积组中的 3-15
adding to Apple menu 添加至苹果菜单的 3-68至3-69
adding to Help menu 添加至帮助菜单的 3-67至3-68
adding to menus 添加至菜单的 3-64至3-70
appearance of 的外观 3-12至3-14
changing 改变
font style of 的字形 3-60
icon of 的图标 3-62至3-64
mark of 的标记 3-61至3-62
script code of 的语系代码 3-62
text of 的文字 3-59至3-60
characteristics of 的特征 3-12至3-14
disabling 停用 3-8, 3-58至3-59, 3-74至3-77
divider 的分隔线 3-12
enabling 启用 3-8, 3-58至3-59, 3-74至3-77
font style of 的字形 3-47, 3-154
grouping 分组 3-14至3-16
handling user selection of 处理用户选定的 3-70至3-78

icon of　的图标 3-154
item numbers　的编号 3-40
keyboard equivalents for　的等效组合键 3-16至3-19, 3-154
marking character of　标记…的特征 3-154
metacharacters in　中的元字符 3-65
in a mutually exclusive group　互斥组中的 3-14
providing help balloons for　为…提供旁述 3-7
script code of　的语系代码 3-47, 3-62, 3-154
script of　的语系 3-12
specifying the characteristics of　指定…的特征 3-45至3-47
submenu for　的子菜单 3-47, 3-154
terms to use in help balloons and user documentation　在旁述和用户文稿中使用的…术语 3-8
text of　的文字 3-12至3-13
MenuKey function 3-　MenuKey 函数 3-71, 3-77至3-78, 3-117至3-118
menu lists　菜单列表 3-40, 3-97至3-98
Menu Manager　菜单管理器 3-5至3-167
application-defined routines for　应用程序定义的…例程 3-148至3-151
data structures in　中的数据类型 3-95至3-102
data structures maintained by　保留的数据类型 3-40至3-41
and Dialog Manager　和对话管理器 6-68至6-73
initializing　初始化 3-103至3-104
localization guidelines for　的本地化准则 3-10, 3-16至3-18, 3-43
routine names, mapping of　的例程名称，的例程名称的映射 3-102
routines in　中的例程　3-102至3-147
testing for availability　测试…的可用性 3-42
user interface guidelines for　的用户界面准则 3-5至3-38
menu records　菜单记录 3-40, 3-95至3-97
menu resource　菜单资源　3-151至3-154
'MENU' resource type　'MENU' 资源类型 3-42至3-48, 3-152至3-154
menus　菜单。参见苹果菜单；应用程序菜单；编辑菜单；文件菜单；字体菜单；帮助菜单；键盘菜单；大小菜单；样式菜单
accessing from alert and dialog boxes　从示警框和对话框存取菜单 3-84至3-86, 6-68至6-73
adding items to　向菜单添加项目 3-64至3-70
adjusting items in　调整菜单中的项目 3-73至3-77
Apple　苹果菜单。参见苹果菜单
Application　应用程序。参见应用程序菜单
color　彩色菜单　3-11
creating　创建菜单 3-42至3-57, 3-105至3-108
hierarchical　创建嵌套式菜单 3-53至3-56
pop-up　创建弹出式菜单 3-56至3-57, 5-25至5-28, 6-42至6-44
pull-down　下拉式菜单 3-42至3-56
disabling for alert and dialog boxes　停用示警框和对话框中的菜单 6-68至6-73
disposing of　处理菜单 3-112, 3-140
Edit　编辑菜单。参见编辑菜单
File　文件菜单。参见文件菜单
Font　字体菜单。参见字样菜单
grouping items in　把菜单中的项目分组 3-14至3-16
Help　帮助菜单。参见帮助菜单
help balloons for　菜单的旁述 3-31
hierarchical　嵌套菜单 3-6, 3-38, 3-53至3-56
items in　菜单中的项目。参见菜单项目
Keyboard　键盘菜单。参见键盘菜单
keyboard equivalents for　菜单的等效组合键 3-16至3-19, 3-154
localizing　菜单本地化 3-10
menu　菜单 ID 3-152
pop-up　弹出式菜单。参见弹出式菜单
pull-down　下拉式菜单 3-6, 3-42
resource ID　菜单的资源标识符 3-44, 3-153
Size　大小菜单。参见大小菜单
Style　式样菜单。参见样式菜单
submenus　子菜单 3-6, 3-53至3-56
system-handled　系统处理的菜单 3-19
titles of　菜单标题 3-7, 3-10
types of　菜单的类型 3-6
MenuSelect function　MenuSelect 函数 3-8, 3-70, 3-72, 3-73, 3-78至3-79, 3-115至3-117
message string resources　消息字符串资源 7-27至7-30
minor switches　次切换 2-16

missing-application name string resources 未找到应用程序名字字符串资源 7-27至7-30，7-68

modal dialog boxes 模式对话框

- accessing menus from 从…中存取菜单 6-68至6-73
- closing 关闭 6-100至6-101
- creating 创建 6-23至6-26，6-113至6-119
- defined 定义 6-10至6-11
- displaying 显示 6-61，6-62至6-75
- event filter function for 的事件过滤函数 6-86至6-89；6-145至6-147
- events in 中的事件 6-82至6-86，6-135至6-137
- window types for 的窗口类型 4-9

ModalDialog procedure ModalDialog 过程 6-84，6-135至6-137

- and disk-inserted events 和插入磁盘事件 2-56
- and menus 和菜单 3-86

modeless dialog boxes 无模式对话框

- accessing menus from 从…中存取菜单 6-68至6-73
- closing 关闭 6-100至6-101
- creating 创建 6-23至6-26，6-113至6-119
- defined 定义 6-12
- displaying 显示 6-61至6-67
- events in 中的事件 6-89至6-100，6-138至6-141
- titles for 的标题 6-25
- window types for 的窗口类型 4-10

modifier keys 修改键 2-20至2-21，2-81，3-77

mounted folder,alias type for 安装的文件夹及其替身类型 7-40

mouse 鼠标

- determining location of 确定鼠标的位置 2-108
- getting information about 获取鼠标的信息 2-108

mouse-down events 鼠标按下事件

- in alert boxes 示警框中的 6-78，6-81至6-82
- in controls 控制部件中的 5-11至5-13，5-30至5-70
- in desk accessories 桌面附件中的 2-87，2-95
- in dialog boxes 对话框中的 6-78至6-80，6-82至6-94
- handling 处理 2-33至2-38
- in the menu bar 菜单条中的 3-72至3-73
- in windows 窗口中的 4-42至4-45

mouse driver 鼠标驱动程序 2-62

mouse-moved events 移动鼠标事件 2-23，2-62至2-67，2-86

mouse region 鼠标区域 2-23，2-64至2-67

mouse-up events 释放鼠标事件 2-33

- in alert boxes 示警框中的 6-78，6-81至6-82
- in controls 控制部件中的 5-11至5-13，5-30至5-70
- in dialog boxes 对话框中的 6-78至6-80，6-82至6-94

movableDBoxProc window type movableDBoxProc 窗口类型 3-85，4-9，6-25

movable modal dialog boxes 可移动模式对话框

- accessing menus from 从…中存取菜单 6-68至6-73
- closing 关闭 6-100至6-101
- creating 创建 6-23至6-26，6-113至6-119
- defined 定义 6-11至6-12
- displaying 显示 6-61，6-62
- events in …中的事件 6-89至6-100，6-138至6-141
- titles for …的标题 6-25
- window types for …的窗口类型 4-9

movable resources 可移动资源 7-36至7-38

MoveControl procedure MoveControl 过程 5-67至5-70，5-97至5-98

MoveWindow procedure MoveWindow 过程 4-95至4-96

MultiFinder 2-7

MyAction procedure MyAction 过程 5-116至5-117

MyAlertSound procedure MyAlertSound 过程 6-144至6-145

MyControl function MyControl 函数 5-110至5-115

MyEventFilter function MyEventFilter 函数 6-145至6-147

MyIndicatorAction procedure MyIndicatorAction 过程 5-117

MyItem procedure MyItem 过程 6-143至6-144

N

nameLocked flag nameLocked 标志 7-18，7-34

name string resources for applications 应用程序的名称字符串资源 7-27至7-30

networks, using shared Trash directories on

网络，在网络上使用共享的废纸篓目录 7-44
NewCDialog function NewCDialog 函数。参见 NewColorDialog 函数
NewColorDialog function NewColorDialog 函数 6-115至6-118
New command (File menu) “新文稿”命令（文件菜单）3-23
NewControl function 5- NewControl 函数 5-82至5-85
NewCWindow function NewCWindow 函数 4-79至4-82
NewDialog function NewDialog 函数 6-118至6-119
NewMenu function NewMenu 函数 3-105至3-106
NewWindow function NewWindow 函数 4-82至4-85
noGrowDocProc window type noGrowDocProc 窗口类型 4-10，6-25
NoteAlert function NoteAlert 函数 6-110
note alerts “注意”示警框
creating with the NoteAlert function NoteAlert 函数创建“注意”示警框 6-110
defined 定义“注意”示警框 6-8
Notification Manager, and operating-system events 通知管理器和操作系统事件 2-12
null events 空事件
defined 定义空事件 2-9
and event masking 空事件和事件屏蔽参数 2-28
handling 处理空事件 2-57

O

OldContent global variable OldContent 全局变量 4-117
OldStructure global variable OldStructure 全局变量 4-117
onlyBackground flag onlyBackground 标志 2-117
Open command (File menu) “打开”命令（文件菜单）3-23
OpenDeskAcc function OpenDeskAcc 函数 3-21，3-80至3-81
Open Documents events 打开文稿事件 7-20，7-25至7-26
opening 打开
applications, from the Finder 打开应用程序，从Finder 中——应用程序 7-25至7-26
disks 打开磁盘 7-20
documents 打开文稿 7-25至7-26
folders 打开文件夹 7-20
Operating System Event Manager 操作系统事件管理器 2-3，2-7。参见事件管理器
Operating System event queue 操作系统事件队列
defined 定义操作系统事件队列 2-6
flushing events from 从操作系统事件队列中删除事件 2-93
operating-system events 操作系统事件
defined 定义操作系统事件 -10至2-12
handling 处理操作系统事件 2-58至2-67
Option key Option键 2-20
OSEventAvail function OSEventAvail 函数 2-98至2-99
Other command (Size menu) “其它”命令（大小菜单） 3-28至3-29，3-82至3-84

P

page regions, in scroll bars 滚卷条中的页面区域 。参见灰度区域
Page Setup command (File menu) “页面设置”命令（文件菜单） 3-23
PaintBehind procedure PaintBehind 过程 4-118至4-119
PaintOne procedure PaintOne 过程 4-118
PaintWhite global variable PaintWhite 全局变量 4-118，4-119
Palette Manager, and system color tables 调色板管理器和系统颜色表 4-21
ParamText procedure ParamText 过程 6-47，6-129至6-130
part codes 部位代码 5-31至5-32
partitions, sizes of （存储器）分区及其大小 2-30，2-115
Paste command (Edit menu) “贴”命令（编辑菜单）
and dialog boxes 和对话框 6-69
and information windows 和信息窗口 7-17
introduced 引入的 3-25
PBGetCatInfo function PBGetCatInfo 函数 7-34，7-49
PBMountVol function PBMountVol 函数 2-55
pCDeskPat parameter-RAM bit flag pCDeskPat 参数—RAM 位标志 4-113

pictures, in alert or dialog boxes 示警框和对话框中的图片 -29至6-30, 6-154
PinRect function PinRect 函数 4-99
plainDBoxProc window type plainDBoxProc 窗口类型 4-9
pop-up menus 弹出式菜单 3-33至3-38
action procedure for 的动作过程 5-36至5-37
active 启用的 5-11至5-13
control definition function for 的控制定义函数 3-33, 3-34至3-36, 5-14
control definition ID for 的控制定义标识符 5-14
creating 创建 3-56至3-57, 5-25至5-28
defined 定义 3-6, 5-6至5-7
in dialog boxes 对话框中的 6-42至6-44
events in, in windows 中的事件, 窗口的…中的事件 5-31至5-37
inactive 停用 5-11至5-13
testing for availability 测试…的可用性 3-42
type-in fields 的键入域 3-37
variation codes for 的变量代码 5-27至5-28
PopUpMenuSelect function PopUpMenuSelect 函数 3-57, 3-120至3-121
popupPrivateData data type popupPrivateData 数据类型 5-77
port names 端口名称 2-75
converting至process serial numbers 转换成进程序号 2-105
and receiving high-level events 和接收高级事件 2-73
port rectangle 端口矩形 4-18, 4-101, 4-103
of Window Manager port 窗口管理器端口的—— 4-75
PostEvent function PostEvent 函数 2-104至2-105
PostHighLevelEvent function PostHighLevelEvent 函数 2-73至2-77, 2-101至2-103
posting options 投寄选项 2-75
PPCBrowser function PPCBrowser 函数
filling in a target ID record 填写一个目标标识符记录 2-77
finding a specific port 查找一个指定的端口 2-75
posting a high-level event 投寄一个高级事件 2-74至2-75
PPC (Program-to-Program Communications) Toolbox PPC (程序间通信) 工具箱 1-16, 2-7
PPostEvent function PPostEvent 函数 2-103至2-104
preferences files 预置选择文件 7-43
Preferences folder 预置选择文件夹
alias type for 的替身类型 7-40
defined 定义 7-43
finding 查找 7-54
storing files in 在…中储存文件 7-43
'pref' folder type 'pref'文件夹类型 7-54
Print command (File menu) "打印"命令 (文件菜单) 3-23
Print Documents events 打印文稿事件 7-25至7-26
PrintMonitor Documents folder 打印监控程序文稿文件夹
alias type for 的替身类型 7-40
defined 定义 7-44
finding 查找 7-54
'prnt' folder type 'prnt'文件夹类型 7-54
processes 进程
background 后台进程 2-4
context of 进程的内容 2-15
foreground 前台进程 2-4
switching between 进程间的切换 2-5, 2-15
Process Manager 进程管理器
and Event Manager 和事件管理器 2-3
and the 和 inder 7-25
process serial numbers, converting至port names 进程序号转换成端口名称 2-105至2-107
Program-to-Program Communications (PPC) Toolbox 程序间通信 (PPC) 工具箱 1-16, 2-7
Publisher Options command (Edit menu) "出版选项"命令 (编辑菜单) 3-25
pull-down menus 下拉式菜单
creating 创建 3-43至3-52
defined 定义的 3-6

Q

QHdr data type QHdr 数据类型 2-83
query documents 询问文稿
creating icons for 为…创建图标 7-16, 7-17
default icon for 的默认图标 7-12
defined 定义 7-4
QuickDraw 1-14 QuickDraw 1-14。参见 彩色 QuickDraw
coordinate systems in QuickDraw 的坐

标系 4-17至4-18
Quit command (File menu) “退出”命令（文件菜单） 3-23

R

radio buttons 单选按钮
active 有效 5-11至5-13
control definition ID for 的控制定义标识符 5-14
creating, in windows 创建，在窗口中创建 5-20至5-21
defined 定义 5-6
events in 中的事件 6-78
in windows 窗口的…中的事件 5-31至5-32
inactive 无效 5-11至5-13
part code for 的部位代码 5-32
specifying in alert or dialog boxes 指定示警框和对话框中的 6-153
rDocProc window type rDocProc 窗口类型 4-10
rectangles, for controls 矩形，控制部件的窗口类型 5-18至5-19, 5-25至5-26, 5-82, 5-118。亦请参
refCon field 阅显示矩形 refCon 域 4-111
reference values 参考值
changing, for controls 改变…，改变控制部件的… 5-105至5-106
determining, for controls 确定…，确定控制部件的… 5-105
for pop-up menus 弹出式菜单的… 5-28
specifying, for controls 指定…，指定控制部件的… 5-18, 5-25, 5-84, 5-120
region codes, in version resources 版本请求中的区域代码 7-31
requiresSwitchLaunch flag requiresSwitchLaunch 标志 7-48
Rescued Items from volume name folders 来自卷名文件夹的拯救项目 7-44
ResetAlertStage procedure ResetAlertStage 过程 6-112
ResetAlrtStage procedure ResetAlrtStage 过程。参见 ResetAlertStage 过程。
resizing windows 调整窗口大小 4-57至4-59
ResolveAliasFile function ResolveAliasFile 函数 7-52至7-53
resource forks 资源分支 1-11
resource IDs 资源标识符
for cursors 光标的 2-63
defined 定义的 1-11
for icons in menu items 菜单项目的图符的 3-62
Resource Manager 资源管理器 1-11至1-13
and Control Manager 和控制管理器 5-3
and the 和 inder 7-3
resource maps 资源映象 1-12
resources 资源 1-11至1-13
alert 警告资源 6-19至6-22, 6-150至6-151
alert color table 警告颜色表资源 6-157至6-158
alias record 替身记录 7-40
application-missing message string 应用程序未找到消息字符串资源 7-68至7-69
bundle 束资源 7-20至7-25, 7-65至7-68
color icon 彩色图标资源 3-62, 7-64
control 控制部件资源 5-18至5-28, 5-118至5-121
control color table 控制部件颜色表 5-121至5-123
control definition function 控制部件定义函数 5-14, 5-109至5-115
defined 定义资源 1-11
dialog 对话资源 6-15, 6-24至6-26, 6-148至6-149
dialog color table 对话颜色表资源 6-75至6-76, 6-156至6-157
driver 驱动程序资源 3-20, 3-68, 3-129, 3-130
file reference 文件参考资源 7-18至7-25, 7-64至7-65
folder 文件夹资源 -55
font 字体资源 3-69
font family 字库资源 -69
icon 图标资源 3-62, 7-63
icon list 图标列表资源 7-13, 7-57至7-58
international configuration 国际配置资源 3-32
item color table 项目颜色表资源 6-158至6-164
item list 项目列表资源。参见项目列表资源
keyboard-layout 键盘布局资源 2-39
key-map 键映象 2-39, 2-41, 2-42
key-remap 键重映象 2-39, 2-41

large 8-bit color icon 大8位彩色图标资源 7-61至7-62
large 4-bit color icon 大4-位彩色图标资源7-59至7-60
menu 菜单资源 3-151至3-154
menu bar 菜单条资源 3-155
menu bar definition function 菜单条定义函数 3-9，3-87，3-104
menu color information table 菜单颜色信息表 3-155至3-157
menu definition procedure 菜单定义过程 3-157
missing-application name string 未找到应用程序名字字符串 7-68
movable 可移动资源 7-36至7-38
signature 署名资源 7-8至7-10
size 大小资源 2-115至2-119，7-35
small 8-bit color icon 小8位彩色图标资源7-62至7-63
small 4-bit color icon 小4-位彩色图标资源 7-60至7-61
small icon 小图标资源 3-62
small icon list 小图标列表资源 7-58至7-59
sound 语音资源 7-37至7-38
string 字符串资源 3-60
string list 字符串列表资源 3-60，3-65至3-67
System file，in the 系统文件资源，系统文件中的资源 1-12
used by the Finder Finder使用的资源7-6至7-32
version 版本资源 7-31至7-32，7-69至7-70
window 窗口资源 -22，4-25至4-27，4-124至4-127
window color table 窗口颜色表资源 4-127至4-129
window definition function 窗口定义函数资源 4-22，4-120，4-127
resource types 资源类型 1-11
'actb' 'actb'资源 -157至6-158
'alis' 'alis'资源7-40
'ALRT' 'ALRT'资源。参见警告资源
'BNDL' 'BNDL'资源 7-20至7-25，7-65至7-68
'cctb' 'cctb'资源5-121至5-123
'CDEF' 'CDEF'资源 5-14，5-109至5-115
'cicn' 'cicn'资源3-62，3-154，7-64
'CNTL' 'CNTL'资源 5-18至5-28，5-118至5-121
'dctb' 'dctb'资源6-156至6-157
'DITL' 'DITL'资源。参见项目列表资源
'DLOG' 'DLOG'资源参见对话资源
'DRVR' 'DRVR'资源 -20，3-68，3-129，3-130
'FKEY' 'FKEY'资源 2-39，3-118
'fld#' 'fld#'资源7-55
'FOND' 'FOND'资源 3-69，3-129，3-130
'FONT' 'FONT'资源 3-69，3-129，3-130
'FREF' 'FREF'资源 -18至7-25，7-64至7-65
'hmmu' 'hmmu'资源 -44，3-68
'icl8' 'icl8'资源 -61至7-62
'icl4' 'icl4'资源7-59至7-60
'ICN#' 'ICN#'资源 7-20，7-57至7-58
'ICON' 'ICON'资源 3-62，3-154，7-63
'ics8' 'ics8'资源7-62至7-63
'ics4' 'ics4'资源7-60至7-61
'ics#' 'ics#'资源7-15，7-58至7-59
'ictb' 'ictb'资源6-158至6-164
'itlc' 'itlc'资源3-32
'itlk' 'itlk'资源2-41
'KCHR' 'KCHR'资源 2-39至2-42
'KMAP' 'KMAP'资源 2-39至2-42
'MBAR' 'MBAR'资源 3-155
'MBDF' 'MBDF'资源 3-9，3-87，3-104
'mctb' 'mctb'资源 -155至3-157
'MDEF' 'MDEF'资源 3-157
'MENU' 'MENU'资源 3-151至3-154
'SICN' 'SICN'资源 3-62，3-154
'SIZE' 'SIZE'资源 2-115至2-119，7-35
'STR ' 'STR '资源 3-60，7-27至7-30
'STR#' 'STR#'资源 3-60，3-65至3-67
'vers' 'vers'资源7-31至7-32，7-69至7-70
'wctb' 'wctb'资源 4-127至4-129
'WDEF' 'WDEF'资源 4-22，4-120，4-127
'WIND' 'WIND'资源 -22，4-25至4-27，4-124至4-127
resume events 恢复事件
defined 定义 2-10

handling 处理 2-60至2-62
ResumeProc global variable ResumeProc 全局变量 6-172
return receipts 回执 2-75，2-77
RGBColor data type RGBColor 数据类型 5-79

S

sample code 示例代码。参见示例例程
sample routines 示例例程
DisplayMyDialog DisplayMyDialog 6-58
DoActivate 2-53，4-51至4-52
DoActivateGlobalChangesDialog 6-98
DoCloseCmd 4-60至4-61，6-94
DoContentClick 5-33，5-53，6-92
DoCopyResource 7-29
DoDiskEvent DoDiskEven 2-56
DoDrumRollCheckBox 5-38
DoEvent DoEvent 2-26，3-72，3-77
DoGlobalChangesDialog DoGlobalChangesDialog 6-67
DoGraphicsScroll DoGraphicsScroll 5-47
DoGrowWindow 4-58至4-59
DoHighLevelEvent 2-70
DoIdle 2-57，6-79
DoKeyDown DoKeyDown 2-44，3-78，6-95
DoMenuCommand DoMenuCommand 3-79
DoMouseDown 2-34，3-72，4-44至4-45，5-32，6-91
DoNew DoNew 5-24
DoNewCmd 4-28至4-29
DoOpenFile 4-37至4-38
DoOSEvent DoOSEvent 2-59
DoPlayButton 5-36
DoPopUpMenu DoPopUpMenu 5-37
DoSaveAsCmd DoSaveAsCmd 7-28
DoShowModelessFindDialogBox 4-64
DoSpellBoxWithSpanish 6-54
DoSuspendResumeEvent 2-61
DoUpdate 2-50，4-50，5-29，5-62，6-99
DoZoomWindow 4-55至4-56
IsStationeryDoc 7-36
MyAddHelpItem 3-68
MyAdjustCursor 2-65
MyAdjustEditMenuForDocWindow 3-75
MyAdjustEditMenuForModalDialogs
MyAdjustEditMenuForModalDialog 6-72
MyAdjustFileMenuForDocWindow 3-74
MyAdjustHV 5-41
MyAdjustMenus MyAdjustMenu 3-73，6-70
MyAdjustMenusForDialogs MyAdjustMenuForDialog 6-70
MyAdjustScrollBars 5-39
MyAdjustScrollSizes 5-67
MyAdjustScrollValues 5-40
MyAlert 6-66
MyAlertSound 6-22
MyChange MyChang enuBarAndSaveColorInfo 3-52
MyCloseDocument 4-61至4-62，6-47
MyCreatePlaySoundsWindow 5-17
MyDrawDefaultButtonOutline 6-59
MyDrawGraphicsWindow 5-63
MyDrawWindow 4-39，5-30，5-65
MyEventFilter 6-88
MyEventLoop 2-24
MyGetLocalUpdateRgn 4-60
MyGetWindowType 4-24
MyHandleAppleCommand 3-80
MyHandleHelpCommand 3-81
MyHandleKeyDown MyHandleKeyDow 2-44，6-95
MyHandleKeyDownInModeless 6-96
MyHandleModelessDialogs 6-49
MyHandleSiz MyHandleSi Command 3-83
MyHorzntlActionProc 5-59
MyMakeAllMenus MyMakeAllMenu 3-70
MyMakeMenus 3-50
MyMakeSub MyMakeSun enu 3-55
MyMDEF 3-89
MyMoveScrollBox 5-61
MyOpen 7-41
MyPostTest 2-74
MyPostWithPPCBrowser 2-76
MyResizeWindow 4-59
MySaveWindowPosition 4-34至4-35
MySetWindowPosition 4-36至4-37
MySpellCheckDialog 6-83
MyToggleHideShow 3-59
MyVerticalActionProc 5-59
ShowMyAboutBox 6-65
UserDidCancel 2-46

Save As command (File menu) “存储为”命令（文件菜单）3-23
Save command (File menu) “存储”命令（文件菜单）3-23
SaveOld procedure SaveOld 过程 4-117
SaveUpdate global variable SaveUpdate 全局变量 4-118
SaveVisRgn global variable SaveVisRgn 全局变量 4-106
scrap 剪贴区
  converting global至private 全局剪贴区到私有剪贴区的转换 2-12
  converting private至global 私有剪贴区到全局剪贴区的转换 2-12
scrap coercion 强制剪贴区 2-58，2-61
Scrap Manager 剪贴区管理器 1-14，2-60
'scri' file type ‘scri’文件类型 7-44
script，changing a menu item's 改变菜单项目的语系 3-62，3-154
script code，for menu item text 菜单项目文字的语系代码 3-47，3-62
script system resource collections，icon for 语系系统资源集合及其图标 7-37
scroll arrows 滚动箭头
  action procedures for 的动作过程 5-58至5-61
  defined 定义的 5-8至5-9
  events in 中的事件 5-57至5-61
  part codes for 的部位代码 5-32
scroll bars 滚卷条。参见灰度区域；滚动箭头；滚动框
  activating 激活 5-13
  active 有效 5-11至5-13
  changing settings and display of 改变…的设定值和显示 5-9至5-10，5-39至5-43，5-44，5-61，5-70
  control definition ID for 的部件标识符 5-14
  creating，in windows 创建…，在窗口中创建 5-21至5-25
  defined 定义 5-7至5-10
  document scrolling with 用…滚动文稿 5-9至5-10，5-43至5-70
  events in 中的事件 5-31至5-32，5-43至5-65
  inactive 无效 5-11至5-13
  and List Manager 和列表管理器 5-3
  moving and resizing 移动…及调整…的大小 5-65至5-70
  updating 更新 5-30
  as a window element 和窗口元素 4-6
scroll boxes 滚动框
  action procedure for 的动作过程 5-36
  defined 定义 5-7至5-8
  dragging 拖拽 5-35
  events in 中的事件 5-53至5-57
  moving 移动 5-9至5-10，5-39至5-43，5-44，5-61，5-70
  part code for 的部位代码 5-32
ScrollRect procedure ScrollRect 过程 5-46至5-48，5-56至5-57
Select All command (Edit menu) “全选”（编辑菜单） 3-25
SelectDialogItemText procedure SelectDialogItemText 过程 6-131至6-132
SelectWindow procedure SelectWindow 过程 2-51，4-87至4-88
SelIText procedure SelIText 过程。参见SelectDialogItemText过程
SendBehind procedure SendBehind 过程 4-91
servers，alias type for 服务器及其替身类型 7-40
session reference number 对话项目参考号 2-73
SetControlAction procedure SetControlAction过程 5-102
SetControlColor procedure SetControlColor 过程 5-101
SetControlMaximum procedure SetControlMaximum 过程 5-41至5-43，5-70，5-95至5-96
SetControlMinimum procedure SetControlMinimum 过程 5-95
SetControlReference procedure SetControlReference 过程 5-105至5-106
SetControlTitle procedure SetControlTitle过程 5-96
SetControlValue procedure SetControlValue 过程 5-38至5-39，5-42至5-43，5-44，5-61，5-94至5-95
SetCRefCon procedure SetCRcon 过程。参见SetControlReference 过程
SetCTitle procedure SetCTitle 过程。参见SetControlTitle 过程
SetCtlAction procedure SetCtlAction 过程。参见SetControlAction 过程
SetCtlColor procedure SetCtlColor 过程。参见SetControlColor过程
SetCtlMax procedure SetCtlMax 过程。参见SetControlMaximum 过程

SetCtlMin procedure SetCtlMin 过程。参见 SetControlMinimum 过程
SetDAFont procedure SetDAFont 过程。参见 SetDialogFont 过程
SetDeskCPat procedure SetDeskCPat 过程 4-112 至 4-113
SetDialogFont procedure SetDialogFont 过程 6-105
SetDialogItem procedure SetDialogItem 过程 6-122 至 6-123
SetDialogItemText procedure SetDialogItemText 过程 6-131
SetDItem procedure SetDItem 过程。参见 SetDialogItem 过程
SetEventMask procedure SetEventMask 过程 2-99 至 2-100
SetItemCmd procedure SetItemCmd 过程 3-139 至 3-140
SetItemIcon procedure SetItemIcon 过程 3-62 至 3-64，3-137 至 3-138
SetItemMark procedure SetItemMark 过程 3-61 至 3-62，3-135 至 3-136
SetItem procedure SetItem 过程。参见 SetMenuItemText 过程
SetItemStyle procedure SetItemStyle 过程 3-60，3-134
SetIText procedure SetIText 过程。参见 SetDialogItemText 过程
SetMCEntries procedure SetMCEntries 过程 3-146 至 3-147
SetMCInfo procedure SetMCInfo 过程 3-52，3-144
SetMenuBar procedure SetMenuBar 过程 3-50 至 3-51，3-112 至 3-113
SetMenuFlash procedure SetMenuFlash 过程 3-142
SetMenuItemText procedure SetMenuItemText 过程 3-59 至 3-60，3-133
SetOrigin procedure SetOrigin 过程 5-44 至 5-45，5-63 至 5-64
SetPort procedure SetPort 过程 6-62
SetWinColor procedure SetWinColor 过程 4-114 至 4-115
SetWindowPic procedure SetWindowPic 过程 4-110
SetWRefCon procedure SetWRefCon 过程 4-111
SetWTitle procedure SetWTitle 过程 4-85
'sfil' file type 'sfil'文件类型 7-37
shared folder, alias type for 共享文件夹及其替身类型 7-40
Shift key Shift 键 2-20
ShortenDITL procedure ShortenDITL 过程 6-127 至 6-128
Show Balloons command (Help menu) "显示旁述"命令（帮助菜单） 6-68
Show Clipboard command (Edit menu) 显示"剪贴板"（编辑菜单） 3-25
ShowControl procedure ShowControl 过程 5-39 至 5-40，5-66 至 5-67，5-86
ShowDialogItem procedure ShowDialogItem 过程 6-124
ShowDItem procedure ShowDItem 过程。参见 ShowDialogItem 过程
ShowHide procedure ShowHide 过程 4-89 至 4-90
ShowWindow procedure ShowWindow 过程 4-88，6-61
'SICN' resource type, specifying a menu item's 'SICN' 资源类型，指定菜单项目的 'SICN' 资源类型 3-62，3-154
signature resources 署名资源 7-8 至 7-10
signatures 署名
  in bundle resources 束资源中的署名 7-20
  as creators 作为创建者的署名 7-9
  defined 定义署名 7-8
  registering 登记署名 7-8 至 7-10
16-by-16 pixel (small) icons 16X16 像素（小）图标 7-11，7-15
size box 调整大小格 4-6
SizeControl procedure SizeControl 过程 5-67 至 5-70，5-98
Size menu 大小菜单 3-27 至 3-29，3-82
  handling 处理 3-82 至 3-84
  Other command in 中的"其它"命令 3-28 至 3-29
size region 调整区域 4-12
size resources 规模资源 2-115 至 2-119，7-35
'SIZE' resource type 'SIZE' 资源类型 2-115 至 2-119，7-35
  creating 创建 2-30 至 2-32
  defined 定义 2-30 至 2-32，2-115 至 2-119
  flags, defined 标志，定义的…标志 2-116 至 2-119
  and null events 和空事件 2-9，2-57
  sample Rez input 的示例Rez 输入格式 2-31
  scheduling option flags 计划…的可选标志

2-16
setting flags for high-level events 设定高级事件的…标志 2-68
setting flags of 设定…的标志 2-9, 2-11, 2-16, 2-51
SizeWindow procedure SizeWindow 过程 4-60, 4-101
sleep value 睡眠值 2-22
slider controls 滑块控制功能 5-11
small 8-bit color icon resources 小 8 位彩色图标资源
creating 创建 7-11, 7-24
Rez output format for 的Rez 输出格式 7-62至7-63
small 4-bit color icon resources 小 4- 位彩色图标资源
creating 创建 7-11, 7-24
Rez output format for 的Rez 输出格式 7-60至7-61
small icon list resources 小图标列表资源
creating 创建 7-11, 7-15, 7-24
Rez output format for 的Rez 输出格式 7-58至7-59
small (16-by-16 pixel) icons 小（16X16 像素）图标
resource IDs for 的资源标识符 7-15
resources for 的资源 7-10
sound resources 语音资源
icon for 的图标 7-37至7-38
installing and removing 安装和删除 7-41
'srvr' alias type 'srvr'替身类型 7-40
stack, size of 堆栈，堆栈的大小 2-32
standard control definition function 标准控制定义函数 5-14
standard file dialog boxes, and disk-inserted events 标准文件对话框，标准文件对话框和插入磁盘事件 2-55
Standard File Package, and disk-inserted events 标准文件软件包，标准文件软件包和插入磁盘事件 2-55至2-56
standard state of a window 窗口的标准状态 4-33, 4-53, 4-70
Startup Items folder 启动项目文件夹
alias type for 的替身类型 7-40
defined 定义 7-44
finding 查找 7-54
static text items 静态文本项目
getting text from 从…中获取文字 6-130至6-131
setting text in 在…中设定文字 6-36至6-41, 6-131, 6-153
specifying in alert or dialog boxes 指定示警框和对话框中的 6-29至6-30, 6-153
substituting text in 替换…中的文字 6-46至6-48
substituting text strings in 替换…中的文字字符串 6-129至6-130
user interface guidelines for 的用户界面准则 6-40至6-41
stationery pads 固定样板文稿
creating icons for 创建…的图标 7-16, 7-19
default icon for 的默认图标 7-12
defined 定义 7-4, 7-34至7-36
recognition of 分辨 2-119
StillDown function StillDown 函数 2-109
StopAlert function StopAlert 函数 6-109
stop alerts "停止"示警框
creating with the StopAlert function 用 StopAlert 函数创建 6-109
defined 定义的 6-9
'STR ' resource type 'STR ' 资源类型 3-60, 7-27至7-30
'STR#' resource type 'STR#' 资源类型 3-60, 3-65至3-67
'strt' folder type 'strt'文件夹类型 7-54
structure region 结构区域 4-6, 4-12
Style data type Style 数据类型 3-134
StyleItem data type StyleItem 数据类型 3-134
Style menu 字形菜单 3-15
example of 的实例 3-15
and the Font menu 字形和字样菜单 3-27
keyboard equivalents for 字的等效组合键 3-19
styles 样式
changing a menu item's 改变菜单项目的样式 3-47, 3-60
of a menu item 菜单项目的样式 3-12
submenus 子菜单
creating 创建 3-53至3-56
defined 定义 3-6
Subscriber Options command (Edit menu) "订户选项"命令（编辑菜单） 3-25
Subscribe 至 command (Edit menu) "SubscribeTO"命令（编辑菜单） 3-25
suitcases for fonts and desk accessories 手提式字形和桌面附件 7-37
suspend events 挂起事件
defined 定义 2-10
handling 处理 2-60至2-62

switching 切换
  context 切换内容 2-15
  major 主切换 2-16
  minor 次切换 2-16
system alert sounds 系统示警声 6-8，6-22
SystemClick procedure SystemClick 过程 2-87，2-94至2-95，4-43，4-44
system color tables，and Palette Manager 系统颜色表，系统颜色表和调色板管理器 4-21
SystemEdit function SystemEdit 函数 3-122
SystemEvent function SystemEvent 函数 2-96
system event masks 系统事件屏蔽参数 2-28至2-29，2-99
system extensions 系统功能扩展
  and Extensions folder 和功能扩展文件夹 7-44
installing and removing 安装和删除 7-41
System file 系统文件 7-44
System Folder 系统文件夹
  alias type for 的替身类型 7-40
  defined 定义 7-5
  finding 查找 7-54
organization of 系统文件的组织 7-41至7-45
system-handled menus 系统处理的菜单 3-19
SystemMenu procedure SystemMenu 过程 3-121
SystemTask procedure SystemTask 过程 2-25，2-87，2-95至2-96

T

TargetID data type TargetID 数据类型 2-72，2-81
target ID records 目标标识符记录 2-81至2-82
  receiving 接收 2-73
  sending 发送 2-75
'temp' folder type 'temp'文件夹类型 7-54
temporary files 临时文件 7-43，7-44
Temporary Items folder 临时项目文件夹
  defined 定义 7-43
  finding 查找 7-54
TEPinScroll procedure TEPinScroll 过程 5-55至5-56
TestControl function TestControl 函数 5-93
TEUpdate procedure TEUpdate 过程 5-64至5-65
text 文字
  in buttons，checkboxes，and radio buttons 按钮、复选框和单选按钮中的文字 5-18，5-19，5-19至5-20，5-25，6-37至6-40，6-153
  editable，in dialog boxes 可编辑文字，对话框中的可编辑文字。参见可编辑文字项目
  handling 处理文字 1-14至1-15
  static，in alert and dialog boxes 静态文字，示警框和对话框中的静态文字。参见静态文本项目
TextEdit
  and Dialog Manager 和对话管理器 6-79至6-80
  handling text 处理文字 1-14至1-15
  and mouse events 和鼠标事件 2-37
'TEXT' file type 'TEXT'文件类型 7-10
text style table 文本样式表 6-160，6-162至6-164
32-by-32 pixel (large) icons 32X32 像素（大）图标 7-11，7-13
thumb controls 拇指型控制部件。参见滚动框
TickCount function TickCount 函数 2-112
Ticks global variable Ticks 全局变量 2-112
title bar 标题栏 4-5
titles 标题
  of buttons，checkboxes，and radio buttons 按钮、复选框和单选按钮的标题 6-37至6-40，6-153
  changing，for controls 改变标题，改变控制部件的标题 5-96
  determining，for controls 确定标题，确定控制部件的标题 5-104至5-105
  for modeless and movable modal dialog boxes 无模式对话框和可移动模式对话框的标题 6-25
  specifying for controls 指定控制部件的标题 5-18，5-19至5-20，5-25，5-83，5-85，5-120，5-121
Toolbox Event Manager 工具箱事件管理器。参见事件管理器
TopMenuItem global variable TopMenuItem 全局变量 3-151
TrackBox function TrackBox 函数 4-44，4-101至4-102
TrackControl function TrackControl 函数 5-35至5-37，5-44，5-53至5-55，5-58，5-90至5-92，6-78
  and pop-up menus 和弹出式菜单 3-56
TrackGoAway function TrackGoAway 函数 4-44，4-103至4-104

Trash, alias type for 废纸篓及其的替身类型 7-40
Trash directories 废纸篓目录
appearance至users 关于…的外观 7-42
defined 定义 7-44
finding 查找 7-54
'trsh' alias type 'trsh'替身类型 7-40
'trsh' folder type 'trsh'文件夹类型 7-54
TrueType fonts TrueType 字样 7-37。参见字样

U

Undo command (Edit menu) "取消"命令(编辑菜单) 3-25, 6-69
UpdateControls procedure UpdateControl 过程 5-86至5-87
UpdateDialog procedure UpdateDialog 过程 6-142至6-143
update events 更新事件 4-13
in alert boxes 示警框中的 6-82, 6-86至6-89
defined 定义 2-9
in dialog boxes 对话框中的 6-85, 6-86至6-89, 6-97至6-100
handling 处理 2-47至2-50, 4-48至4-50
routines for handling 处理…的例程 4-106至4-107
for windows 窗口的 4-41
update region 更新区域
defined 定义 4-13
maintaining 保留 4-41
redrawing 重新绘制 4-40至4-41
UpdtControl procedure UpdtControl 过程。参见UpdateControl 过程
UpdtDialog procedure UpdtDialog 过程。参见UpdateDialog 过程
user interface guidelines 用户界面准则
for Control Manager 控制管理器的 5-52至5-53
for controls 控制部件的 5-5至5-13
for Dialog Manager 对话管理器的 6-8至6-12, 6-31至6-34, 6-37至6-41, 6-42, 6-62至6-64
for icons 图标的 7-13, 7-15
for static text items 静态文本项目的 6-40至6-41
for windows 窗口的 4-5至4-7, 4-21, 4-30至4-32
user-oriented design 面向用户的设计 1-5
user state of a window 窗口的用户状态 4-33, 4-53, 4-70
useTextEditServices flag useTextEditService标志 2-119

V

ValidRect procedure ValidRect 过程 4-58, 4-108至4-109
ValidRgn procedure ValidRgn 过程 4-109
variation codes 变量代码
for controls, defined 控制部件的…, 定义的控制部件的… 5-14
determining, for controls 确定…, 确定控制部件的… 5-106至5-107
for windows 窗口的… 4-11, 4-112, 4-120
version information for applications 应用程序版本信息
in Finder windows Finder 窗口中的 7-31
in information windows 信息窗口中的 7-32
in signature resource 署名资源中的 7-8
in 'vers' resource 'vers'资源中的 7-31至7-32
version resources 版本资源 7-31至7-32, 7-69至7-70
'vers' resource type 'vers'资源类型 7-31至7-32, 7-69至7-70
virtual key codes 模拟键代码 2-40
of Apple Extended Keyboard II, domestic 苹果扩展键盘 II, 美国版 2-43
of Apple Extended Keyboard II, ISO 苹果扩展键盘 II, ISO 版 2-43
of Apple Keyboard II, ISO 苹果键盘 II, ISO 版 2-42
visible region 可视区域 4-48
volume catalogs, Finder information in 卷编目, 卷编目中的Finder 信息 7-32至7-34
volumes, Finder's desktop database for 卷, Finder 的卷编目桌面数据库 7-45

W, X, Y

WaitMouseUp function WaitMouseUp 函数 2-109
WaitNextEvent function WaitNextEvent 函数

2-85至2-88
introduced 引入 2-22至2-24
and multitasking 和多任务 2-60
'wctb' resource type ‘wctb’资源类型 4-71，4-127至4-129
'WDEF' resource type ‘WDEF’资源类型 4-22，4-120，4-127
WinCTab data type WinCTab 数据类型 4-71至4-72
window color table 窗口颜色表 4-21，4-71至4-73
window definition functions 窗口定义函数
defined 定义 4-10
writing 编写 4-120至4-124
window definition IDs 窗口定义标识符
creating windows, used in 创建窗口的…，在创建窗口中使用的… 4-80，4-83至4-84
defined 定义的窗口定义标识符 -11
for a dialog box . 对话框的窗口定义标识符 6-25
and window definition functions 窗口定义函数的窗口定义标识符 4-120
in window resources 窗口资源的窗口定义标识符 4-26，4-125
window frames 窗口框架 4-6，4-12至4-13，4-121至4-122
window list 窗口列表 4-15，4-74
WindowList global variable WindowList 全局变量 4-74
Window Manager 窗口管理器 4-3至4-129
application-definedroutine for 应用程序定义的…例程 4-120至4-124
and Control Manager 和控制管理器 4-14至4-15，5-3
data structures in 中的数据结构 4-65至4-74
and Dialog Manager 和对话管理器 4-13至4-14，6-15至6-16
events generated by 生成的事件 2-9
global variables 全局变量 4-75
port 端口 4-74
resources in 中的资源 4-124至4-129
routines in 的例程 4-74至4-119
initializing 初始化…的例程4-74至4-75
low-level routines 中的低级例程 4-116至4-119
window origin 窗口起始位置
changing 改变 5-44至5-45，5-48至5-52
defined 定义的 5-45
WindowPeek data type WindowPeek 数据类型 4-20，4-65
window positioning constants 窗口位置常数 4-32
WindowPtr data type WindowPtr 数据类型 4-19，4-65
WindowRecord data type WindowRecord 数据类型 4-19，4-65，4-69
window records 窗口记录 4-19至4-20，4-65至4-69
window regions 窗口区域 4-12至4-13
window resources 窗口资源 4-22，4-25至4-27，4-124至4-127
windows 窗口 4-3至4-129
activating 激活窗口 4-50至4-53
active 现用窗口 4-6
alert boxes and dialog boxes 示警框和对话框窗口
deactivating behind 停用示警框和对话框后面的窗口 6-64至6-66
as types of 作为窗口 类型的示警框和对话框 6-15至6-17
closing 关闭窗口 4-60至4-62，4-103至4-106
color in 窗口 的颜色 4-20至4-21，4-71至4-74
content region 窗口 内容区域 4-12
controls in 窗口 中的控制部件 4-14至4-15，5-82
creating 创建窗口 4-25至4-30，4-75至4-85
deallocating 解除分配窗口 4-61至4-62，4-104至4-106
defined 定义窗口 4-4
displaying 显示窗口 4-86至4-91
drawing content region 绘制窗口 的内容区域 4-39至4-40
events in 窗口 中的事件 4-21，4-22，4-41至4-53
grow image 窗口 的增长图象 4-57，4-87
hiding 隐藏窗口 4-62至4-64，4-89
inactive 非当前活动窗口 4-6至4-7
maintaining update region of 保留窗口 的更新区域 4-106至4-109
managing multiple 管理多个窗口 4-15至4-16，4-23至4-25
manipulating 操纵窗口

characteristics 特征 4-109至4-112
color 颜色 4-114至4-116
on the desktop 在桌面上操纵窗口 4-112至4-114
mouse-down events in 窗口 中的按下鼠标事件 4-42至4-45
moving 移动窗口 4-53，4-94至4-99
naming 命名窗口 4-85至4-86
parts drawn by the Window Manager 用窗口管理器绘制窗口部分 4-12至4-13
positioning 窗口定位 4-30至4-38
regions in 窗口中的区域 4-12至4-13
resizing 重新调整窗口 的大小 4-57至4-59
responding to activate events in 响应窗口 中的激活事件 2-50至2-55，4-50至4-53
retrieving information 从窗口中检索信息 4-91至4-94
scrolling 滚动窗口 5-9至5-10，5-43至5-70
showing 显示窗口 4-62至4-64，4-88
sizing 调整窗口 的大小 4-99至4-101
standard state 窗口 的标准状态 4-33，4-53，4-70
structure region 窗口 的结构区域 4-12
types of 窗口 的类型 4-8至4-10
updating 更新窗口 2-47至2-50，4-40至4-41，4-48至4-50，5-49至5-52，5-62至5-65
user state 用户状态 4-33，4-53，4-70
window-manipulation conventions 操纵窗口 的惯例 4-21
window resources，defining 窗口 资源，定义窗口资源 4-25至4-27
zooming 缩放窗口 4-53至4-56，4-101至4-103
window state data record 窗口状态数据记录 4-33，4-54，4-70至4-71
window types 窗口类型 4-8至4-11
'WIND' resource type ‘WIND’ 资源类型 4-22，4-25至4-27，4-124至4-127
wristwatch cursor 手表光标 2-63
WStateData data type WStateData 数据类型 4-33，4-54，4-70至4-71

Z

zoom box 缩放框 4-6
zoomDocProc window type zoomDocProc 窗口类型 4-8
zooming windows 缩放窗口 4-53至4-56，4-101至4-103
zoomNoGrow window type zoomNoGrow 窗口类型 4-10
zoom region 缩放区域 4-12
ZoomWindow procedure ZoomWindow 过程 4-54，4-57，4-102至4-103

责任编辑：徐一帆

INSIDE MACINTOSH 系列丛书

苹果电脑有限公司

INSIDE MACINTOSH 工具箱基础

《Inside Macintosh 工具箱基础》描述了在应用程序中实现基本用户界面元素的方法。Inside Macintosh 工具箱位于Macintosh的心脏，每一位创建Macintosh应用程序的程序员均应熟悉此书提供的资料。

该书特别解释了如何：

- ■创建菜单
- ■创建窗口、对话框及提示框
- ■创建控制，如按钮和滚卷条
- ■为应用程序和文件创建图标
- ■相应用户的动作

当你熟悉了在《Inside Macintosh 概述》中讨论的Macintosh的基本概念、编程环境和直观生动的操作环境后，就应开始阅读此书。

在设计应用程序时，您应参考《Macintosh用户界面设计指南》一书来全面了解Macintosh用户界面的各个方面。

《Inside Macintosh 系列丛书》按主题编写，描述了Macintosh系统软件，为程序员、设计员和工程师提供了基本参考资料。本书封三给出了《Inside Macintosh 系列丛书》的示意图。

ISBN 7-03-004983-7

9 787030 049834 >

ISBN 7-03-004983-7
TP·500
定　价：95.00 元
科技新书目：374-72